United States
Department of
Agriculture

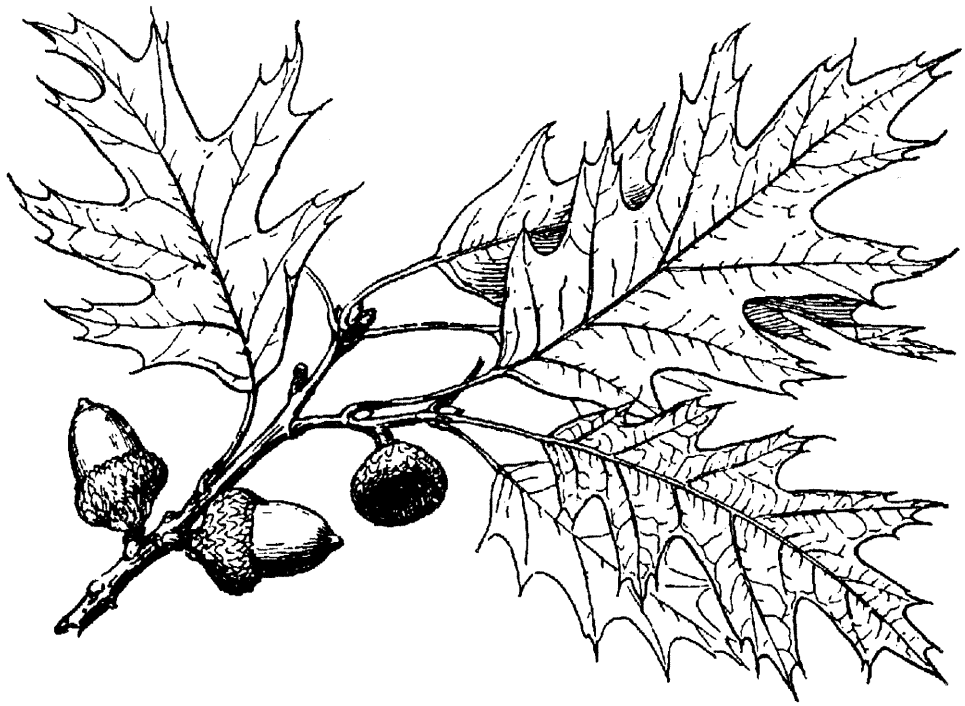
Forest Service

**Northeastern Forest
Experiment Station**

General Technical
Report NE-209

Guide to the Stand-Damage Model Interface Management System

George Racin
J. J. Colbert



Abstract

This programmer's support document describes the Gypsy Moth Stand-Damage Model interface management system. Management of stand-damage data made it necessary to define structures to store data and provide the mechanisms to manipulate these data. The software provides a user-friendly means to manipulate files, graph and manage outputs, and edit input data. The interface was built using pop-up windows, menuing systems, text editing and validation, mouse support, and context-sensitive help. The interface is written in the C language for DOS microcomputers.

The Authors

GEORGE RACIN is a computer specialist with the Northeastern Forest Experiment Station's research laboratory in Morgantown, West Virginia. He received B.S. and M.S. degrees in agricultural engineering from Cornell University and is a licensed professional engineer. He joined the USDA Forest Service in 1989.

J. J. COLBERT has served since 1989 as a research scientist in the silvicultural options for the gypsy moth project at the Northeastern Forest Experiment Station, Morgantown, West Virginia. He received A.B. (1966), M.S. (1970), and Ph.D. (1975) degrees in mathematics. He worked 4 years with the Atomic Energy Commission as a nuclear operations engineer and 3 years as research associate professor at Oregon State University before joining the USDA Forest Service in 1979. He was the research coordinator and then Program Manager of the western component of the Canada/ United States Spruce Budworms Program until 1985.

Manuscript received for publication 13 October 1992

USDA FOREST SERVICE
5 RADNOR CORP CTR SUITE 200
RADNOR PA 19087-4585

August 1995

Contents

Introduction

This support document is meant for programmers who need to maintain and/or modify the Interface Management System for the Gypsy Moth Stand-Damage Model. It also can be used as a guide for those wishing to construct an interface management system.

Computer users expect an interface that reflects recent advances in software standards. An interface should be event driven and provide pull-down menus, pop-up windows, on-line help messages, and mouse support. Using a compiler that supports multi-language development, the C-code interface described here can be used with existing batch-process models by linking it to a model written in another language (such as FORTRAN). This procedure would consist of the following steps:

1) Use the interface to generate input files needed by the existing model.
2) Use the Run Model menu item to call the user's existing model as a subroutine.

Using this procedure, one can link the interface with existing models by making relatively small changes to the interface code itself.

There are two companion documents: "Description of the Stand-Damage Model: Part of the Gypsy Moth Life System Model" (General Technical Report NE-208) describes the model's equations and parameters. The "User's Guide to the Stand-Damage Model: a Component of the Gypsy Moth Life System Model" (General Technical Report NE-207) provides software instructions.

The management of stand-damage data made it necessary to define structures to store data and to provide the mechanisms to manipulate these data. The interface maintains the integrity of default values and allows the user to save and manipulate input data files. The stand interface is written in Microsoft C 6.0 (Microsoft Corp. 1990) for DOS microcomputers.[1] The interface was built using the tools provided in C-scape Version 3.2 (Oakland Group 1989). The C-scape frame menu is used to call all routines necessary to manipulate files, view and manage outputs, and edit input data.

The Stand Model interface provides a user-friendly means to:

- Retrieve default stand data files or input files created and saved from an earlier use of the program.
- Edit the default file or data from other files to create unique model input files.
- Save user input.
- Create temporary stand files that will be used by the Stand Model.
- Access output tables and files.
- Run the model.
- View, graph, print, and save model outputs.

The following terms and conventions are used throughout this document:

- Forest data: Used to define all data- weather, stand management, general, defoliation, etc.
- Stand data: Used to define information specific to a particular stand.
- Species data: Represents individual tree host-species data.
- Stand model: Used to describe the stand-damage submodel of the Gypsy Moth Life System Model. The terms "stand" and "stand damage" are used interchangeably.
- Screen: Collection of characters surrounded by a border placed on the display for some distinct purpose (a "help" screen, a "data entry" screen).
- CAPITAL LETTERS within the code: Boolean variables, constants, and variables used by the FORTRAN stand model consist of all caps.
- `MONOSPACE FONT`: File names are placed in courier font.
- *Italics*: Used for special emphasis.
- **Bold Text**: Functions and variable names are in boldface throughout the text.

Interface Library Overview

C-scape library functions were used for interface development. C-scape is an object-oriented interface development system and has a library of C functions used to create different types of text and data entry screens. These functions are used to create windows, context-sensitive help, text editing and data validation screens, and scrolling lists. C-scape is highly portable: hardware and operating system dependencies are encapsulated in a device-interface structure swappable at run time. C-scape supports many operating systems, including DOS, OS/2, UNIX X-Windows, VMS, and Data General Avion.

The C-scape menu object contains the structure and format of a screen. The menu is defined with the **menu_Printf** function (analogous to C's printf). **Menu_Printf** displays a variable description on the

[1] The use of trade, firm, or corporation names in this paper is for the information and convenience of the reader. Such use does not constitute an official endorsement or approval by the U.S. Department of Agriculture or the Forest Service of any product or service to the exclusion of others that may be suitable.

screen, provides a data field to edit the variable value, and validates the data entered for correctness. **Menu_Open** creates a new menu object. **Menu_flush** releases the memory created by **menu_Printf**.

Once a menu is defined with **menu_Printf**, **sed_Open** creates the "sed" (an object that manages the screen's active display information: Screen EDitor). **Sed_Repaint** paints the sed onto the display. **Sed_Go** gets input from the user until the user decides to exit. **Sed_Close** closes the sed and releases any memory it used.

C-scape provides a standard way to navigate between menus, windows, and data entry fields. C-scape also supports the use of a mouse. When keys are used, the keystrokes are:

- *Escape key.* Exits windows.
- *Arrow keys.* Up and down arrow keys move the cursor between adjacent data entry fields.
- *Enter key.* Moves the cursor to the next data entry field. The Enter key is used to select highlighted framer menu items.
- *First letter of menu items.* Menu items are selected by pressing the first letter of the menu item name.
- *Space bar.* The space bar is used to mark items in a list.
- *Page Up/Page Down.* When text is displayed, Page Up moves to the preceding screen and Page Down moves to the next screen.
- *Home/End.* When a text file is displayed, Home moves to the beginning of the file and End to the bottom of the file. On menus with a list of choices, Home moves to the top of the list and End moves to the bottom of the list.

Disk File Organization

There are six disk files required for proper program execution (the disk file name is shown in parenthesis):

- Default data file (`DEFAULT`): This file is read when the program begins and contains all of the data required to run the Stand-Damage Model.
- Species file (`SPECIES`): This file contains default parameter values for 20 individual host species. When a new tree species is selected, the initial parameters for the new species are read from this file.
- Introduction file (`INTRO`): This file contains a short introductory description of the model and instructions for program menus and keystrokes.
- Help file (`HELP`): This file contains the text of the help messages. It is used to provide context-sensitive help when the F1 function key is pressed.
- Help index file (`HELP.IDX`): Index for help file.
- Graphics file (`OAK.PCX`): Graphic displayed in introductory screen.

Framer Menu Items

The initial screen displays the program title followed by a brief introduction to the program. The C-scape framer menu provides a general outline of the primary sections of the code. The framer menu creates a horizontal menu bar across the top of the screen, and allows a pull-down submenu to be associated with each of the choices. Since the menu bar serves as the top layer for program organization, the nine framer-menu choices provide an overview of the interface (Fig. 1). Only the first level below the framer menu is shown.

Framer Menu	Level 1
SETUP	
	user_edit
	about_damage
FILE	
	save_file
	get_file
	display_inp_files
	reload_default_file
STAND	
	gen_edit
	stand_edit
	stand_data
TREES	
	choose_host_edit
	delete_species
	add_species
	choose_defol_yr_edit
	delete_defol_yr
	add_defol_yr
	remove_all_defol
MANAGE	
	choose_mgmt_yr_edit
	delete_mgmt_yr
	add_mgmt_yr
WEATHER	
	weather_edit
OUTPUT	
	select_outputs
	choose_view_or_print
	choose_output_for_draw
	output_to_save
RUN	
	run_model
QUIT	
	exit_routine

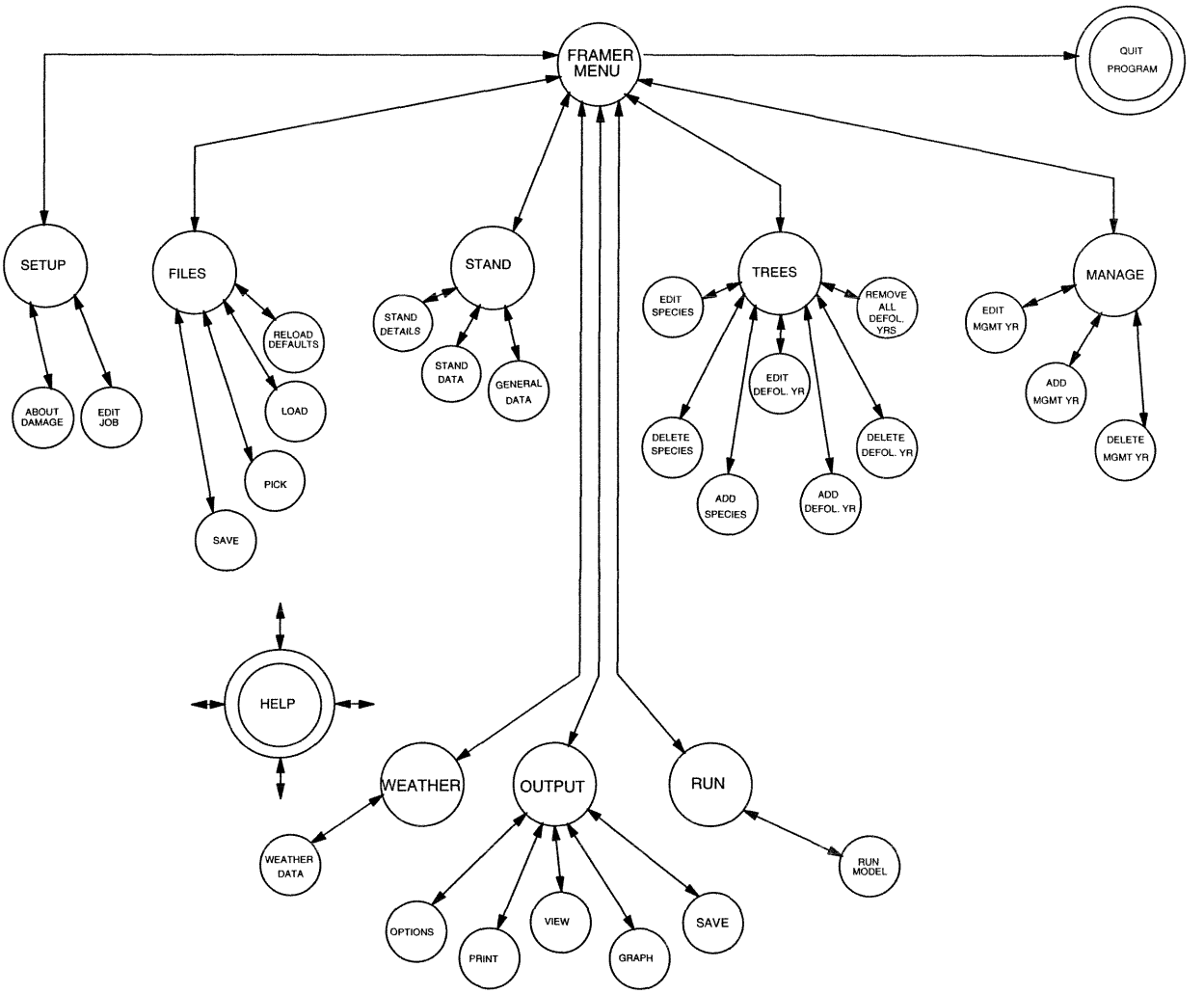


Figure 1. State transition graph for framer menu.

Setup

Setup provides a means by which users can name their disk file and describe their simulation. When the program begins, all description variables are loaded with default values from the disk "default" file. The code consists of one routine called from the C-scape framer menu: **user_edit**.

Framer	Level 1	Level 2
SETUP		
	user_edit	
		extend_file_name
		remove_blanks

User_edit allows the user to edit the disk file name associated with the job. The disk file name is blank when the program begins. **User_edit** also allows the user to enter four short job description fields: user name, job title, site location, and date. The user also can enter a 500-character description of the simulation being run. A C-scape bob, an object embedded within a C-scape sed, was used to embed the area in which to edit the 500-character text buffer.

Remove_blanks removes embedded blanks and periods from the entered file name. **Extend_file_name** concatenates **forestfile_name** and .INP to ensure that all input files have the ".INP" extension. The flow chart that illustrates **user_edit** is shown in Figure 2.

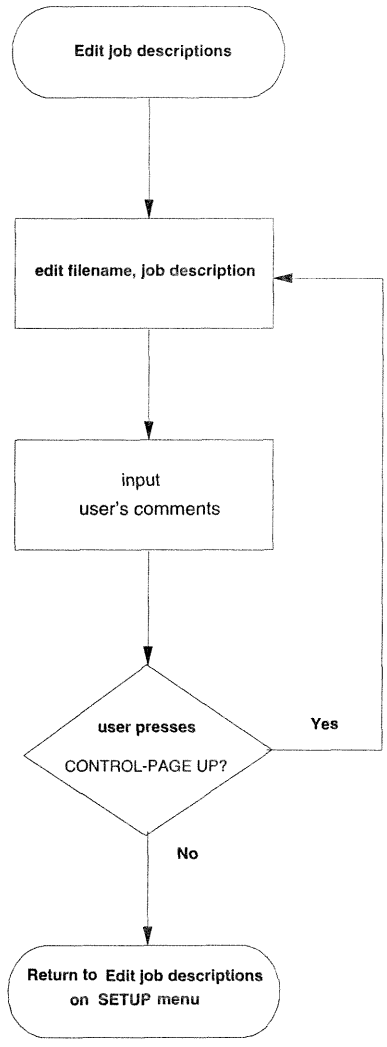


Figure 2. Edit job description.

Framer	Level 1	Level 2	Level 3
FILE			
	save_file		
		forestfile_edit	
			remove_blanks
			extend_file_name
		verify_choice	
		save_forest	
	get_file		
		forestfile_edit	
			remove_blanks
			extend_file_name
		verify_choice	
		read_forest	
	display_inp_files		
		verify_choice	
		read_forest	
	reload_default_file		
		verify_choice	

File

One file is associated with each data set. When the program begins, this file is the default disk file (DEFAULT). Once the user has named a file and saved it, the file will be named "NAME.INP."

The user must open a disk file to retrieve previously developed data. The routines in this section keep track of whether the entered file names exist on disk, whether files may be overwritten, and whether an input data file name has been entered.

The code consists of four routines called from the C-scape framer menu: **save_file**, **get_file**, **choose_inp_file**, and **reload_default_file**.

When File is selected from the framer menu, the pull-down menu choices are Save, Load, Pick, and Reload default file.

Save_file first checks for a blank file name. If a blank name is found, users are prompted with a message indicating that a file name must be entered. **Forestfile_edit** is then called to allow the user to enter a new file name.

If the file name exists on disk, the user is warned that continuing will overwrite existing data. If the user wishes to enter a new file name, **forestfile_edit** is called. If the user does not wish to enter a new file name, the disk file is overwritten with the **save_forest** routine. The flow chart illustrating **save_file** is shown in Figure 3.

The **get_file** routine first checks a flag (**EDITS_MADE**) that indicates whether any editing has been done. If editing has been done, the user is notified so that the routine can be exited without getting a new file (and possibly losing edited data). If editing has not been done, or if the user does not wish to exit **get_file**, **forestfile_edit** is called, enabling the user to enter a new file name. If the entered name does not exist on disk, the user can enter a new name or exit the routine. If the entered name does exist on disk, several steps must be taken before the file is read. First, **read_forest** is called to reset variables with default values. Existing management and defoliation linked lists are destroyed, freeing memory. The file is then opened and the variables are read from disk. The flow chart that illustrates **get_file** is shown in Figure 4.

Choose_inp_file first obtains the input files that have an ".INP" extension in the current directory. The file names are displayed and when the user chooses a file name, the file name is copied to the user's file name. The file is opened and the variables are read from disk with **read_forest**. The flow chart that illustrates **choose_inp_file** is shown in Figure 5.

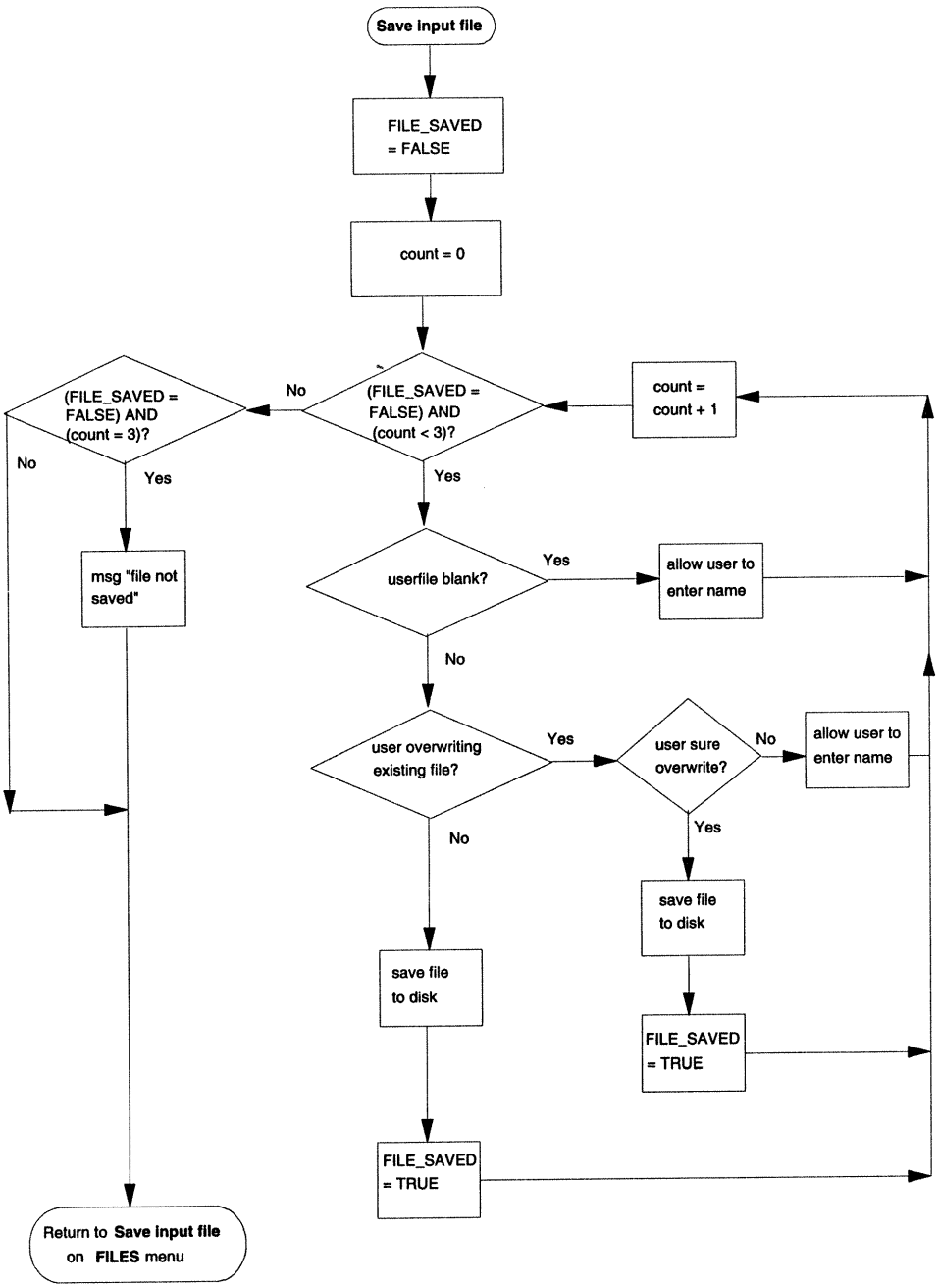


Figure 3. Save file.

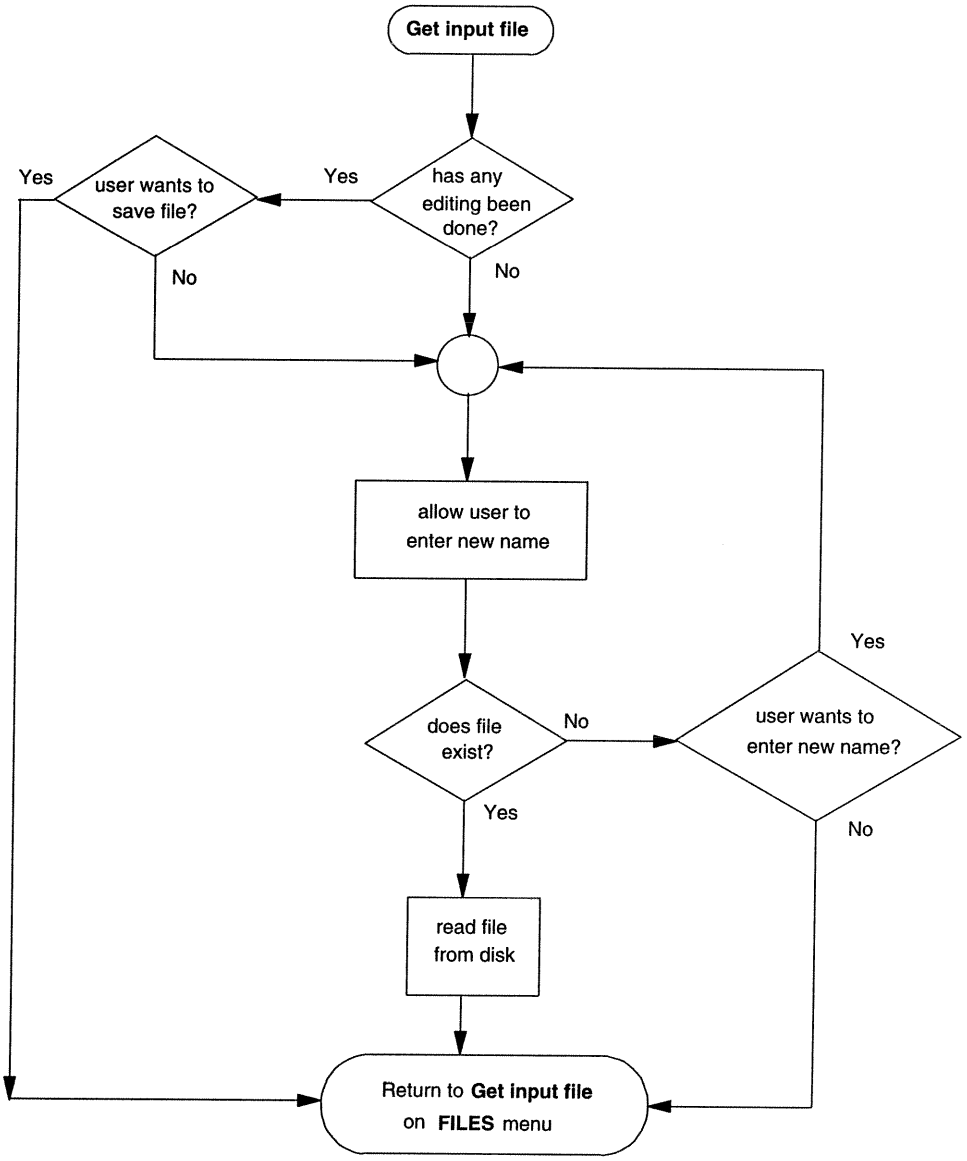


Figure 4. Get file.

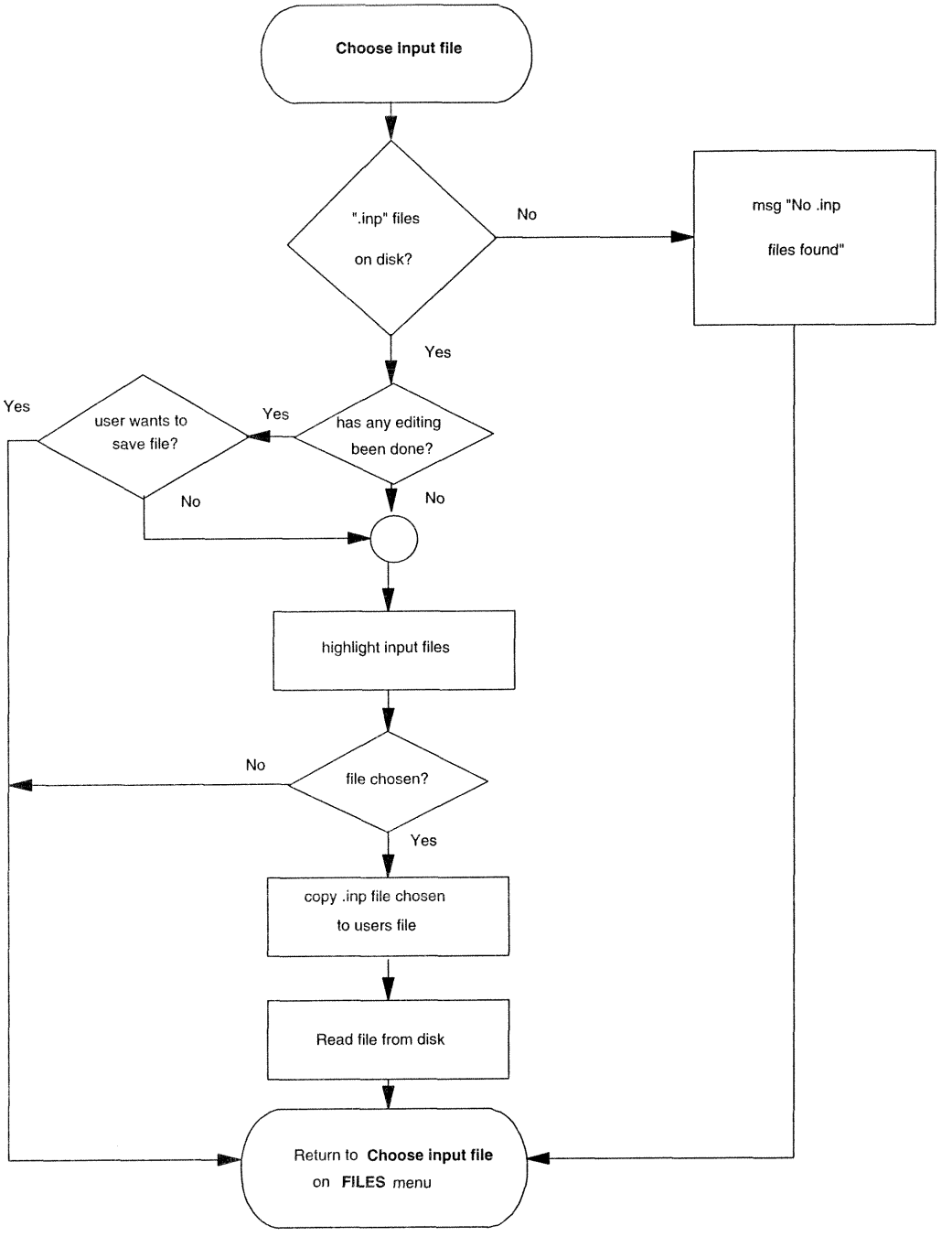


Figure 5. Display input files.

Framer	Level 1	Level 2
STAND		
	gen_edit	
		do_years_agree
		adjust_ISTR_array
	stand_edit	
		choose_ISTR_yrs
	stand_data	

Stand

The stand section allows the user to edit both general (nonbiological) data associated with the simulation and whole-stand data. Two routines are called from the framer menu: **gen_edit** and **stand_edit**.

Gen_edit allows the user to edit general data. If the initial year (**ISYEAR**) or the length of simulation (**NYEAR**) is

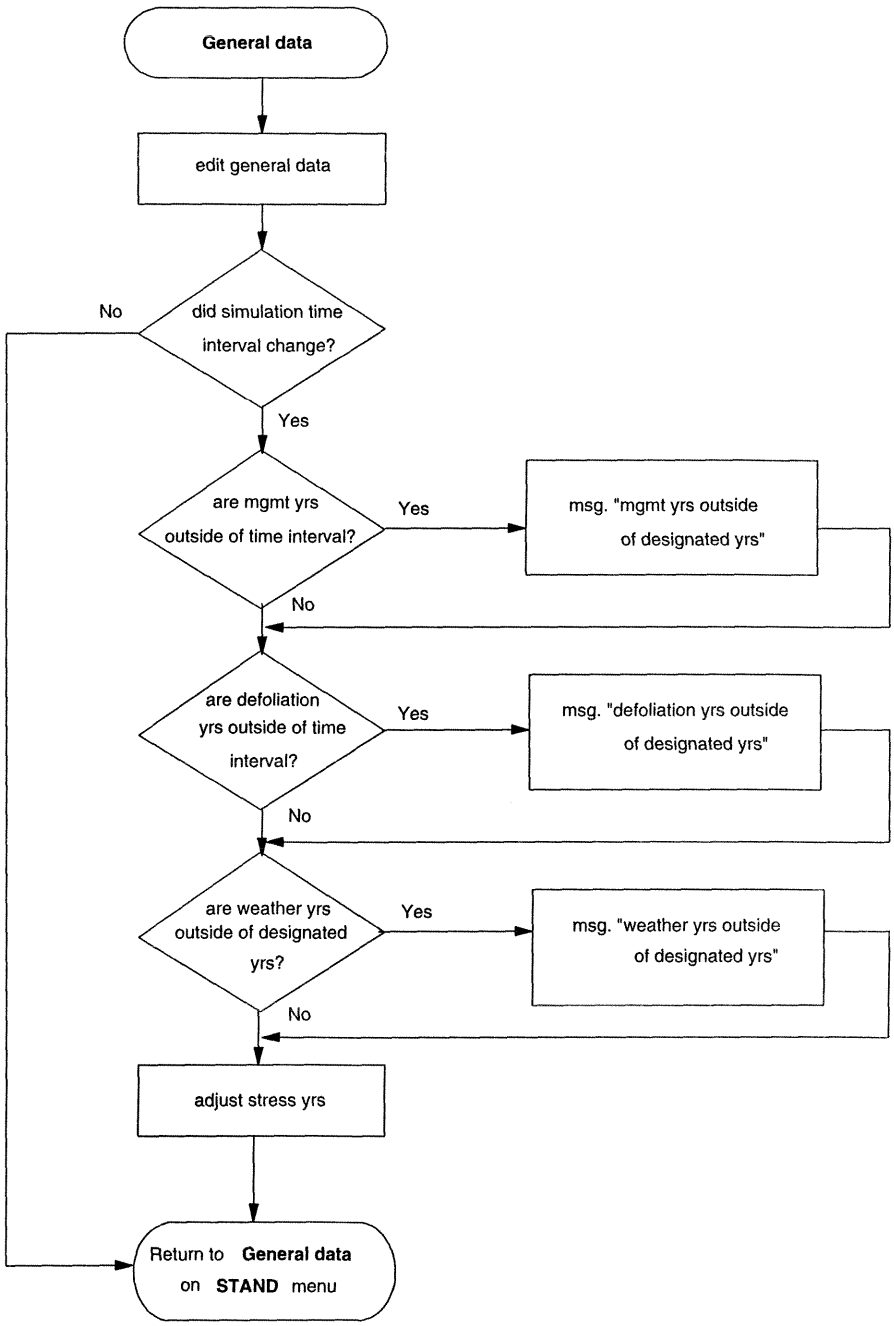


Figure 6. General data.

changed, two routines are called: **do_years_agree** and **adjust_ISTR_array**. In **do_years_agree**, if any existing management, defoliation, or weather years fall outside the new time interval, warning messages are displayed indicating that the appropriate management, defoliation, or weather data may have been made invalid by the time change.

Adjust_ISTR_array changes the array of **ISTR** structures to reflect the change in time interval. This frees the user from having to select stress years when the old and new time intervals overlap. The flow chart illustrating **gen_edit** is shown in Figure 6.

Stand_edit allows the user to edit stand data. When stress option variable **ISOPT** is equal to 2 upon exit from **stand_edit**, a call to **choose_ISTR_yrs** is made. **Choose_ISTR_yrs** displays the available simulation years and allows the user to select individual stress years. The flow chart that illustrates **stand_edit** is shown in Figure 7.

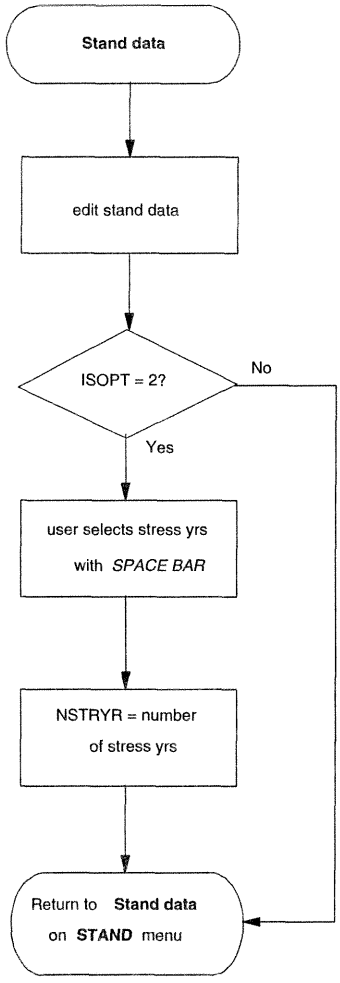


Figure 7. Stand data.

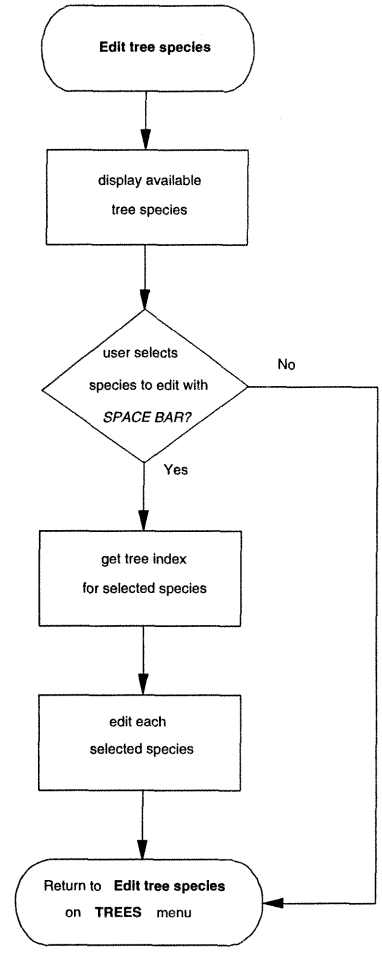


Figure 8. Edit tree species.

Trees

The stand model can simulate from one to MAX_HOSTS host species. The trees section facilitates adding host species, deleting host species, and editing host-species data. At least one host species must be present at all times. There are default parameters for more than 20 tree species. There is a structure for each tree species with an array of structures describing the stand. When a host species is deleted from the structure array, the program deletes the indicated species structure and adjusts the other elements of the array accordingly. When a host species is added to the structure array, the associated parameters are added to the end of each structure array. The "SPECIES" DOS file contains the parameter values for the host species.

The code consists of three routines called from the framer menu: **choose_host_edit**, **delete_species**, and **add_species**.

When Trees is selected from the framer menu, the pull-down menu choices are Choose species, Delete species, and Add species.

Framer	Level 1	Level 2	Level 3
TREES			
	choose_host_edit		
		get_tree_index	
		host_edit	
	delete_species		
		verify_delete	
		delete_host_values	
		get_tree_index	
	add_species		
		open_species_file	
			read_one_species
		host_edit	

The Choose species routine displays the available species to edit. The host editing routine is called for each host species that has been selected. After editing is completed, Escape will exit the trees section and return the user to the framer menu. The flow chart that illustrates **choose_host_edit** is shown in Figure 8.

Upon entry to the Delete species routine, the current species are numbered and displayed. When the user decides to delete a host, the indicated host must be deleted. **Host_to_delete** is assigned the number of the species chosen. If **host_to_delete** is equal to **NHOST**, **NHOST** is decreased by one, eliminating the last host. Otherwise, the function **delete_host_values** is passed **host_to_delete**. **Delete_host_values** shifts all of the species arrays down by one to overwrite the deleted host-species values and reorder the other species-array values. It also reorders the arrays in the stand management and defoliation linked lists. Upon return from **delete_host_values**, **NHOST** is decreased by one. Once a species has been deleted, the species data cannot be recovered. The flow chart illustrating **delete_species** is shown in Figure 9.

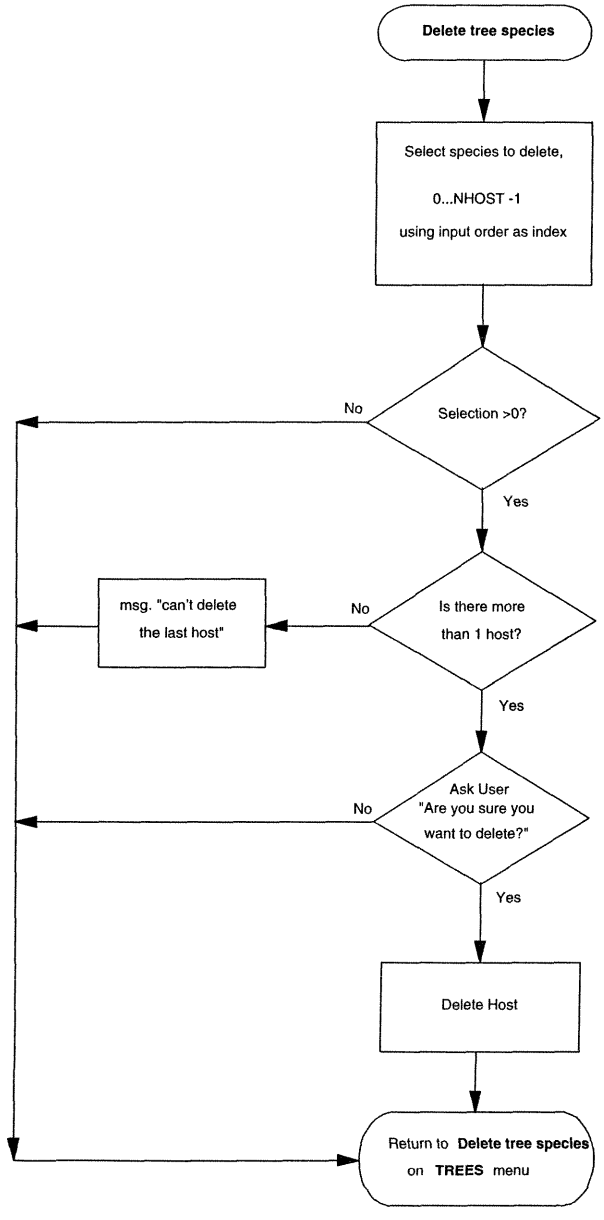


Figure 9. Delete tree species.

Add_species adds a species (**NHOST** + 1). If **NHOST** is already equal to MAX_HOSTS, a message is displayed indicating that another species cannot be added. Otherwise, a list of the available species names is shown. If no selection is made, Escape returns the user to the framer menu.

Open_species_file is called to open the "SPECIES" file containing the host-species values. Once the unwanted species in the species file are skipped and the proper position in the file is reached, **read_one_species** is called. It is called with two parameters: a pointer to the species file and the host index. The pointer to the species file is passed so that the file does not have to be opened within **read_one_species**, making it possible to use **read_one_species** for files with different file formats (for example, the default forest file has species data, but it also has general, stand, stand management, and weather data). The host index is passed so that appropriate array elements (**tree[i].GROWR**, **tree[i].CR1**, etc.) can be assigned values. Upon return from **read_one_species**, the **tree[i].STEMS** array is assigned values. **Tree[i].STEMS[0]** is set to 1 so that the FORTRAN model will have at least one **STEMS** value. The rest are set to 0.

Upon return to **add_species**, **host_edit** is called to enable the user to edit the new host parameters. The flow chart that illustrates **add_species** is shown in Figure 10.

Defoliation

The stand model can make use of up to MAX_DEFOL_YRS (100) years of defoliation. The defoliation section facilitates adding defoliation years, deleting defoliation years, and editing defoliation data. The code consists of four routines called from the C-scape framer menu: **choose_defol_yr_edit**, **add_defol_yr**, **delete_defol_yr**, and **remove_all_defol**.

A structure tracks the parameters associated with a particular defoliation year. Defoliation year and the array defoliation amount (**defol_amt**) are contained in this structure (a two-dimensional array: overstory or understory by host):

Framer	Level 1	Level 2	Level 3
TREES (DEFOLIATION)			
	choose_defol_yr_edit		
		defol_edit_for_one_yr	
			get_tree_index
	delete_defol_yr		
		verify_choice	
		linked_defol_yr_extract	
		linked_defol_yr_delete	
	add_defol_yr		
		initialize_defol_yr_ structure	
		linked_defol_yr_ duplication	
		defol_edit_for_one_yr	
			get_tree_index
		linked_defol_yr_insert	
	remove_all_defol		
		verify_choice	

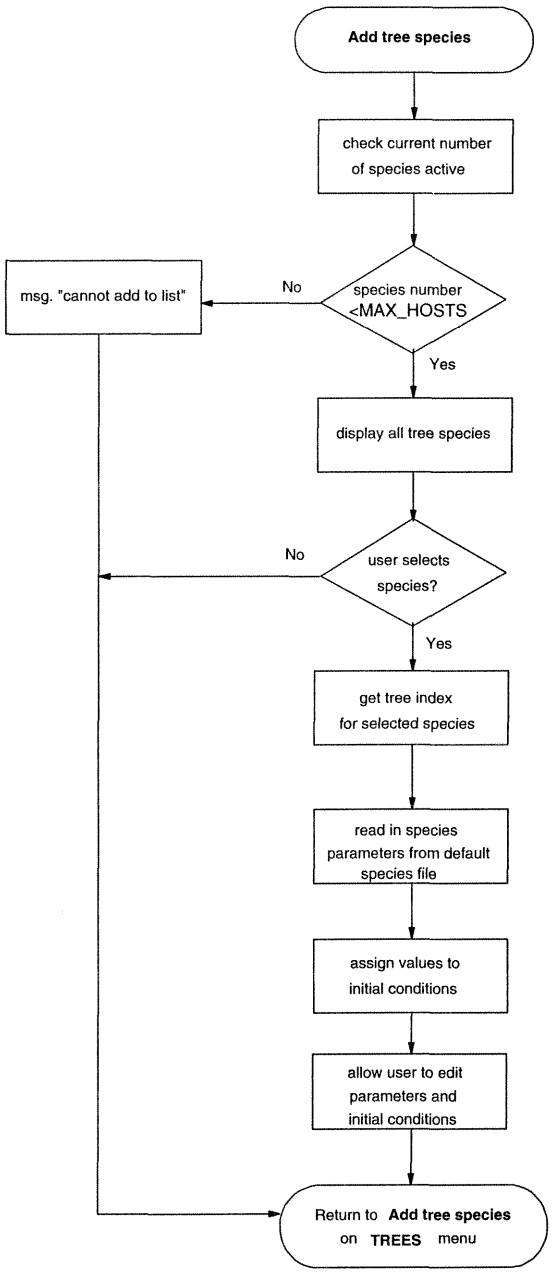


Figure 10. Add tree species.

```
typedef struct defol_str {
  int defol_yr_key;
   int defol_amt[2][MAX_HOSTS];
  struct defol_str *next;
} DEFOL_YRS;
 DEFOL_YRS *start_defol_yr_ptr, defol_yr_struct;
```

Linked-list routines keep track of the defoliation year structures. They are ordered by defoliation years (the linked-list "key"). The linked-list routines are **linked_defol_yr_insert** (inserts structure into linked list, keeping the list in ascending order); **linked_defol_yr_duplication** (checks for duplication of key insertion into linked list); **linked_defol_yr_extract** (extracts structure from linked list; if keyed entry not found, returns error); and **linked_defol_yr_delete** (deletes structure from linked list; if list is empty or if the key not found, error returned).

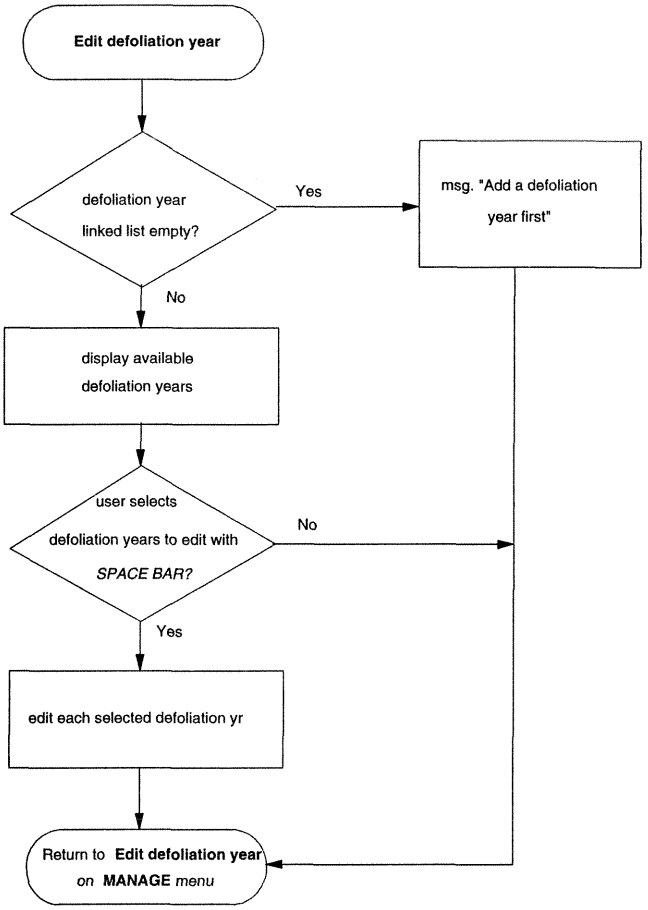


Figure 11. Edit defoliation year.

On the defoliation portion of the Trees framer menu, the pull-down menu choices are Choose defoliation year, Delete defoliation year, Add defoliation year, and Remove all defoliation.

Choose_defol_yr_edit displays the available years to edit. The **defol_edit_for_one_yr** routine (a routine to edit defoliation parameters for a single year) is called for each defoliation year that has been selected. The user edits each of the selected defoliation years. The flow chart that illustrates **choose_defol_yr_edit** is shown in Figure 11.

Upon entry to the Delete defoliation year routine, the **yr_key_to_delete** variable is set to zero. A list of the current years is displayed and the user enters a year from those listed. The entered year is checked to ensure that it is a valid year. If it is, **verify_choice** is called to make sure the user wishes to delete the indicated year. **Linked_defol_yr_delete** is then called with three parameters: **start_defol_yr_ptr** (pointer to front of the list); **yr_key_to_delete** (the key entry of structure to delete); and **status** (a status variable that returns an error

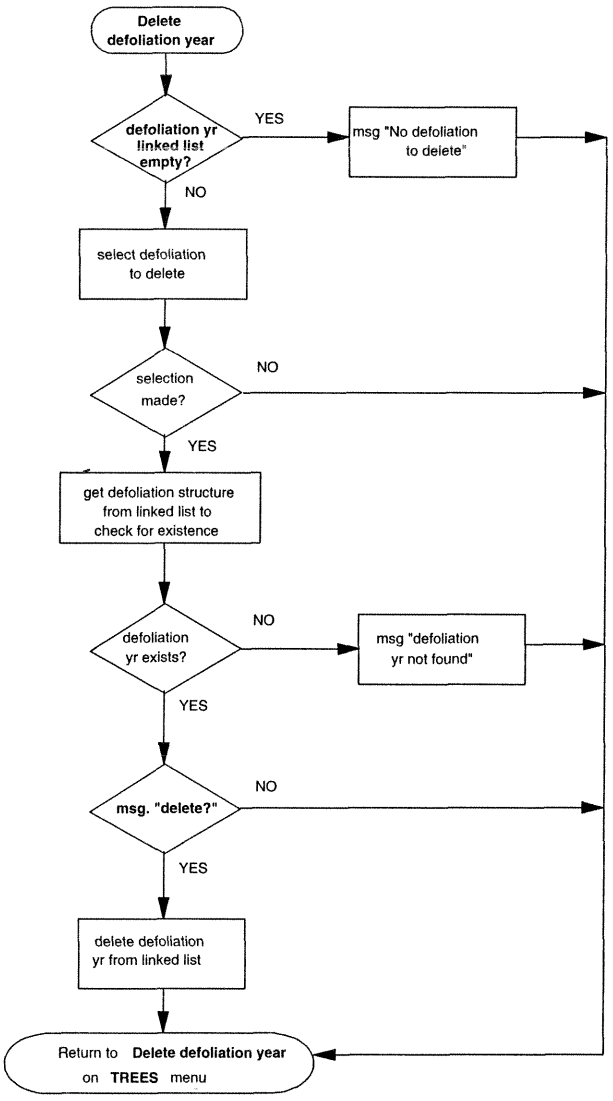


Figure 12. Delete defoliation year.

if the delete was unsuccessful). The **linked_defol_yr_delete** routine returns the pointer to the front of the new list. The flow chart illustrating **delete_defol_yr** is shown in Figure 12.

In **Add_defol_yr** the user enters a new defoliation year. **Linked_defol_yr_duplication** is called to check for duplicated defoliation years. Assuming no duplications, an initialization routine gives the **defol_yr_struct** initial values. Upon return from the initialization routine, a defoliation-year edit routine is called to allow the user to edit the defoliation data. A call is then made to **linked_defol_yr_insert** to insert the edited structure into the linked list. The flow chart that illustrates **add_defol_yr** is shown in Figure 13.

Remove_all_defol deletes all defoliation years from the linked list and sets the number of defoliation years to zero. The flow chart that illustrates **remove_all_defol** is shown in Figure 14.

Stand Management

The stand model can have MAX_MGMT_YRS (15) years of distinct management practices. The stand management section facilitates adding management years, deleting management years, and editing stand management data. The code consists of three routines called from the C-scape framer menu: **choose_mgmt_yr_edit**, **add_mgmt_yr**, and **delete_mgmt_yr**.
The stand management section closely parallels the defoliation section. They use similar routines to add, delete, select, and edit linked-list items. For this reason, details that use identical logic as the defoliation section are omitted.

Framer	Level 1	Level 2	Level 3
MANAGE			
	choose_mgmt_yr_edit		
		mgmt_edit_for_one_yr	
			get_tree_index
	delete_mgmt_yr		
		verify_choice	
		linked_mgmt_yr_extract	
		linked_mgmt_yr_delete	
	add_mgmt_yr		
		initialize_mgmt_yr_ structure	
		linked_mgmt_yr_ duplication	
		mgmt_edit_for_one_yr	
			get_tree_index
		linked_mgmt_yr_insert	

A structure keeps track of the parameters associated with a particular management year. Management year and management method (named **manage**) are items in the structure. The arrays in this structure are **cutmin**, **cutmax**, and **action** (arrays of MAX_HOSTS size):

```
typedef struct mgmt_str {
  int mgmt_yr_key;
  int manage;
  float cutmin[MAX_HOSTS];
  float cutmax[MAX_HOSTS];
  float action[MAX_HOSTS];
  struct mgmt_str *next;
} LINK_YRS;
LINK_YRS *start_mgmt_yr_ptr, mgmt_yr_struct;
```

Linked-list routines keep track of the management year structures that are ordered by management years (the linked-list "key"). These routines are **linked_mgmt_yr_insert** (inserts structure into linked list, keeping list in ascending order); **linked_mgmt_yr_duplication** (checks for duplication of key insertion into linked list); **linked_mgmt_yr_extract** (extracts structure from linked list; if keyed entry not found, returns error); and **linked_mgmt_yr_delete**

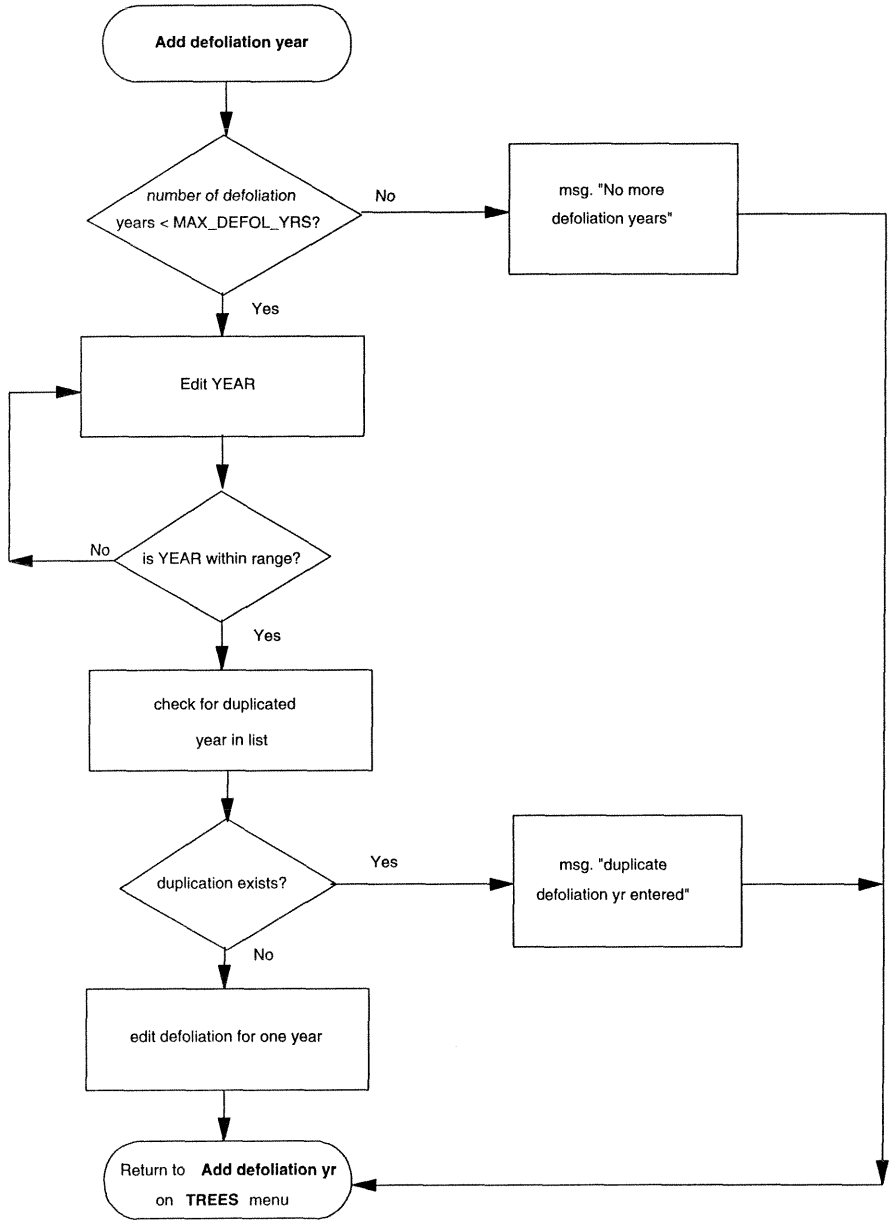


Figure 13. Add defoliation year.

(deletes structure from linked list; if list empty or if key not found, error returned).

When Manage is selected from the framer menu, the pull-down menu choices are Choose management year, Delete management year, and Add management year. **Choose_mgmt_yr_edit** allows the user to choose particular management years to edit. Its operation parallels **choose_defol_yr_edit**. The flow chart illustrating **choose_mgmt_yr_edit** is shown in Figure 15.

Delete_mgmt_yr allows the user to delete stand management years from the linked list. Its operation parallels **delete_defoliation_yr**. The flow chart that illustrates **delete_mgmt_yr** is shown in Figure 16.

Within **Add_mgmt_yr** the user enters the management year and management method (**manage**). Then **linked_mgmt_yr_duplication** is called to check for duplicated management years. When there are no duplications, an initialization routine assigns initial values to **mgmt_yr_struct**. After initialization, a management year edit routine is called. Next, **linked_mgmt_yr_insert** is called to insert the edited structure into the linked list. The flow chart that illustrates **add_mgmt_yr** is shown in Figure 17.

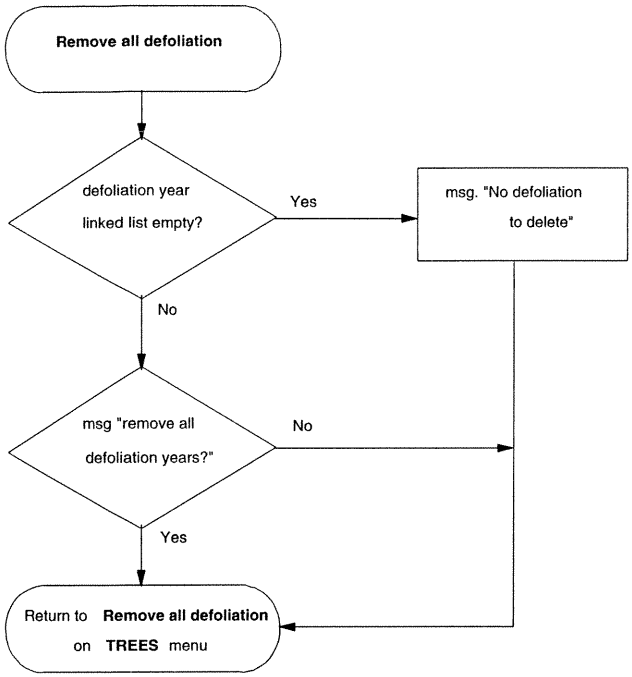


Figure 14. Remove all defoliation.

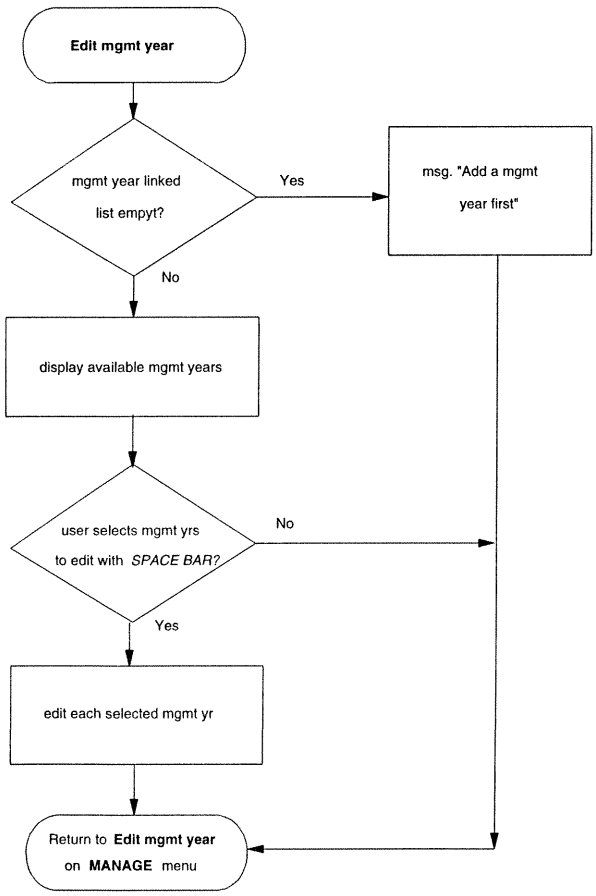


Figure 15. Edit management year.

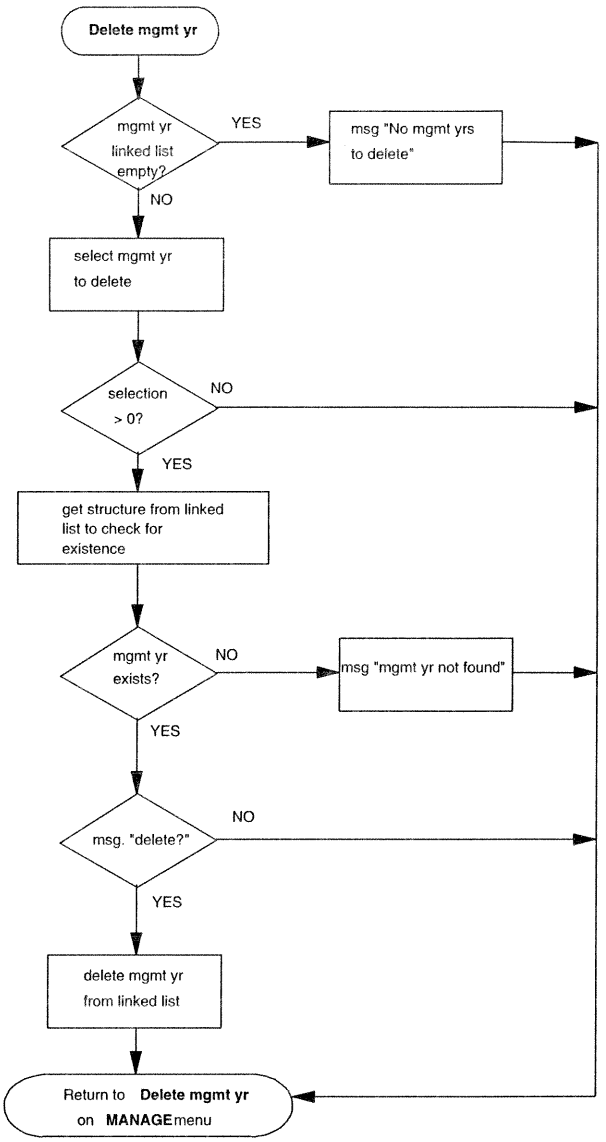


Figure 16. Delete management year.

Weather

The weather section facilitates editing weather data (annual degree days) for each simulation year. The code consists of one routine called from the C-scape framer menu: **weather_edit**.

Model Output

The stand model can produce up to six output files depending on which outputs have been requested. Communication between the FORTRAN model and the C-user interface is through temporary disk files. This section describes the code to view, print, graph, or save each output file.

Each output file has a specified constant file name, i.e., FORTRAN code always opens a predefined file name in

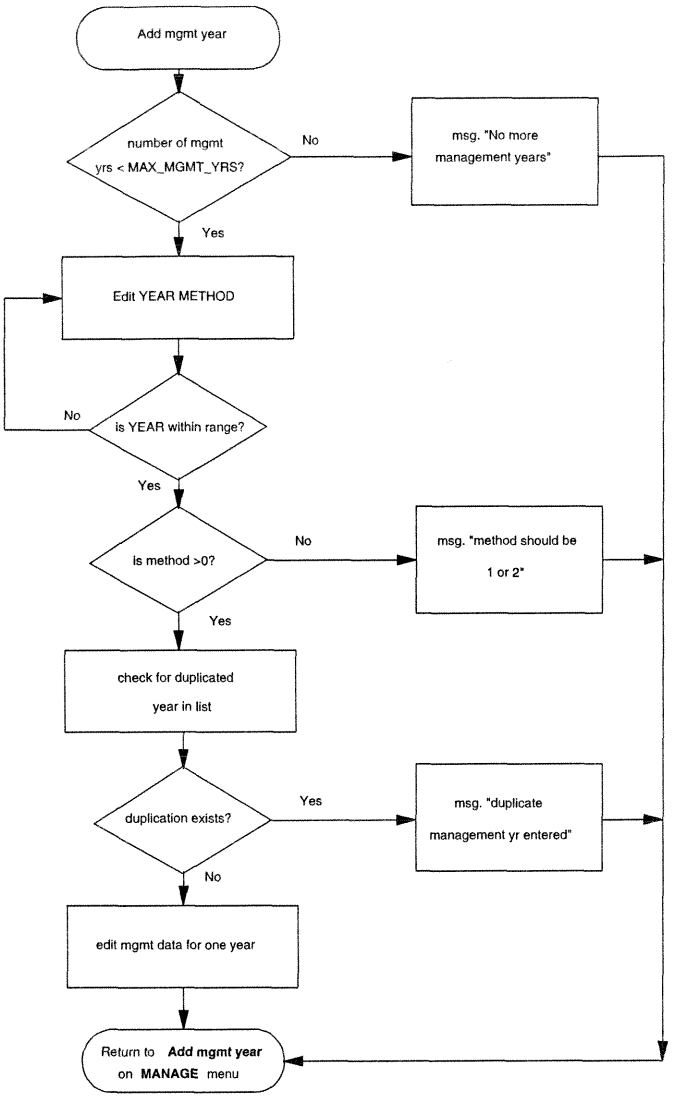


Figure 17. Add management year.

the current default directory and writes the associated output to that temporary file. The user can give the output files unique names (the FORTRAN output files are copied to the user-defined files). The model-output files are erased upon exit from the program or are overwritten by subsequent simulations. The code consists of four routines called from the menu: **select_outputs**, **choose_output_and_view**, **output_to_save**, and **choose_output_for_draw**.

The model can produce six output files. A structure tracks the parameters associated with a particular output file. A six-element array of structures is declared as:

```
struct output_file_str {
    char output_needed[2];
    boolean SAVE_OUTPUT;
    char description[31];
    char model_outputfile_name[13];
    char user_outputfile_name[13]; };
struct output_file_str  outputs[MAX_OUTPUT_FILES];
```

SAVE_OUTPUT is set to TRUE when the model produces an output file. **Model_outputfile_name** is the system file name produced by the FORTRAN model. **User_outputfile_name** is the unique system file name chosen by the user.

When Output is selected from the framer menu, the pull-down menu choices available are Options, View, Save, Print, and Graph.

Select_outputs displays output parameters and allows the user to select outputs. After editing, the variable **IPYEAR** is checked for a value of zero. (**IPYEAR** is the interval in years for printing stand output.) If **IPYEAR** is equal to zero, the routine **IVIEW_edit** is called that allows the user to edit **IVIEW** values (the years that stand table output are to be printed.) The flow chart that illustrates **select_outputs** is shown in Figure 18.

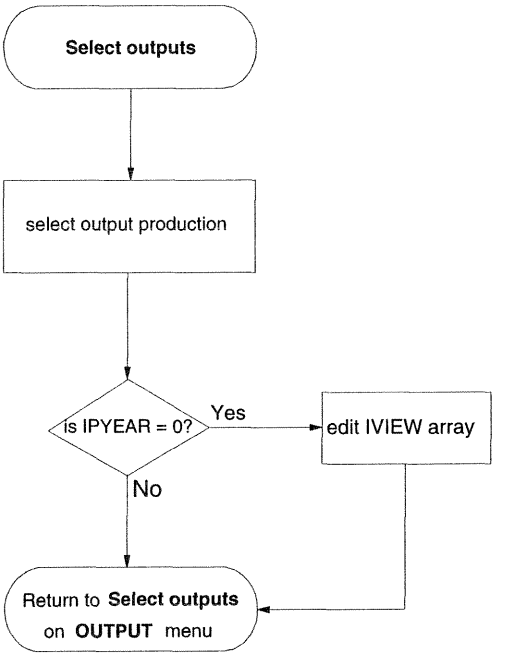


Figure 18. Select outputs.

When View is selected, **choose_view_or_print** calls **view_output**. The current disk directory is searched for model-generated output files. If none are found, a message informs the user that the model must be run first. If output files exist, the titles for existing output files are shown as menu items. If a file is selected, the file is opened and the file is displayed. After the file has been examined, pressing Escape returns control to the framer menu. The flow chart that illustrates **choose_output_and_view** is shown in Figure 19.

When Save is chosen, **output_to_save** is called. The default disk directory is searched for model-generated

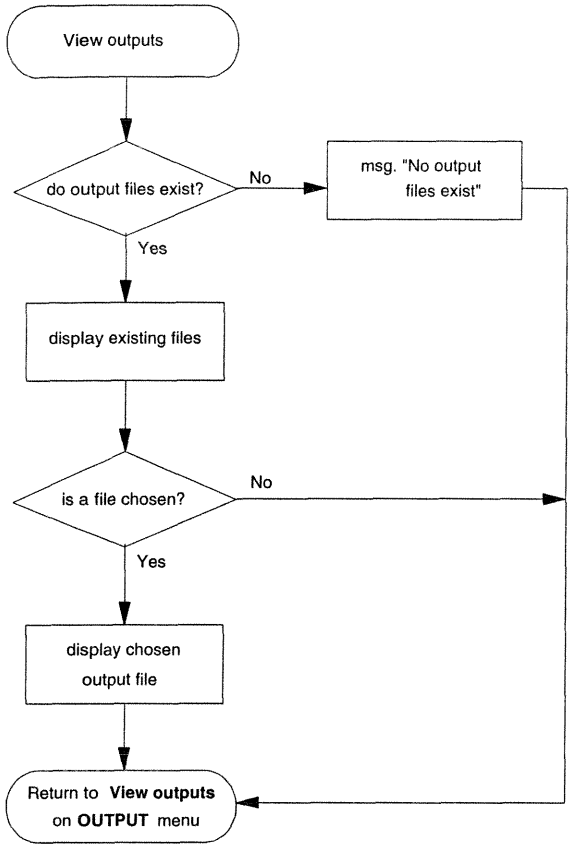


Figure 19. View outputs.

output files. If none are found, a message is displayed informing the user that the model must be run first. If FORTRAN-generated temporary output files exist, output titles are displayed and the user can save the needed files.

Framer	Level 1	Level 2	Level 3
OUTPUT			
	select_outputs		
		IVIEW_edit	
			insertion_sort
			move_array_zeros
			delete_array_dups
	choose_view_or_print		
		view_output	
		print_output	
			printer_test
			copy_file
			verify_choice
	output_to_save		
		create_outputfile_name	
			check_drive_letter
		copy_file	
	choose_output_for_draw		

If a file has been selected, the **model_outputfile_name** file is copied to the **user_outputfile_name** using the **copy_file** routine. The flow chart that illustrates **output_to_save** is shown in Figure 20.

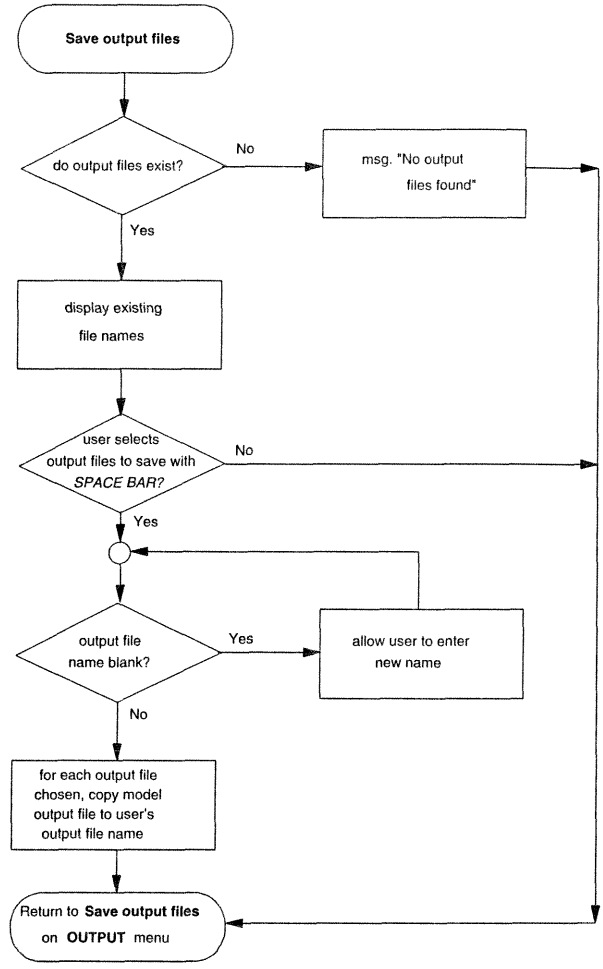


Figure 20. Save output files.

When Print is chosen, **choose_view_or_print** calls **print_output**. **Printer_test** is called to ensure that the printer is functioning properly. **Copy_file** sends the file to the printer.

When Graph is chosen, **control_output_draw** is called by **choose_output_for_draw**.

The graphics portion of the program produces two-dimensional images on the screen from points, straight lines, and character strings. These output primitives are grouped into segments. The windowing transformation maps the objects in world coordinate space (feet, cubic feet/acre, and so on) into viewport coordinates (screen locations). The transformation is performed by setting up a matrix to be applied to each point of the world coordinate definition. The viewport specification is made independent of the particular device used to display the picture.

Level 2	Level 3	Level 4	Level 5
control_output_draw (called by choose_output_for_draw)			
	init_graph_parameters		
	read_graph_NHOSTS		
	read_XYcoord		
	init_graphic_output		
	init_window_viewport		
		set_window	
			set_window_viewport
		set_viewport	
			set_window_viewport
	init_graphics		
		set_video_mode	
		map_colors	
		system_inquire	
		close_graphics	
	XYGraph		
		changetextstyle	
		set_viewport	
			set_window_viewport
		line	
		X_to_screen	
		Y_to_screen	
	close_graphics		

Normalized device coordinates of the view surface are real numbers that range from 0 to 1 in both X and Y, with the origin in the upper left corner of the screen.

When Graph outputs is chosen, **choose_output_for_draw** is called. If a file is selected for viewing, **control_output_draw** is called to graph the output.

Init_graph_parameters is called to set the graphics parameters to zero. **Read_graph_NHOSTS** and **read_XYcoord** retrieves host-species information and graph coordinates from files generated by the model. **Init_graphic_output** generates labels for the graphic image. **Init_window_viewport** finds the minimum and maximum values of X and Y coordinates so that they can be used to define the world-coordinate system (**set_window**). **Set_viewport** defines the screen-coordinate system. Both **set_window** and **set_viewport** call **set_window_viewport** to compute the transformation matrix with the most recent window and viewport parameters.

After returning from **init_window_viewport**, **init_graphics** starts the graphics system. The fonts are read from disk and registered, and the video mode is set to the highest resolution possible (**set_video_mode**). **Map_colors** is called to define a color map for the graph. **System_inquire** is called to obtain the number of pixels available for the monitor.

After returning from **init_graphics**, **XYGraph** draws the image on the screen. **XYGraph** calls **changetextstyle** to set the font height, width, and style. The locations for headings, legends, etc. are then calculated in normalized device coordinates (0 to 1 in X and Y). Once the labels and legend are displayed, **set_viewport** delineates the graphics portion of the screen. To draw the lines of the graph, world coordinates (coordinates in feet, years, and so on) are passed to the **X_to_screen** and **Y_to_screen** functions, which convert the world coordinates to screen coordinates before drawing them on the screen. The flow chart illustrating **choose_output_for_draw** is shown in Figure 21.

Run Model

Run model creates temporary files for the Stand Model, runs the FORTRAN Stand Model, then deletes the temporary Stand Model files. The code consists of one routine called from the framer menu: **run_model**.

Run_model first calls **create_model_files**, which creates the stand-model input files. **Save_forest** is called next with a "`CRASHED.INP`" file name so that if the stand model has a fatal error, the user will not lose any edited data. The stand model (`DAMAGE.OBJ`) is then called. The stand model is written in Microsoft FORTRAN.

After returning from the stand model, the output file's **SAVE_OUTPUT** parameter is toggled on if the appropriate model-output file exists and this file has been requested. Both the stand-model files and the temporary "crash" file are deleted. Although **create_model_files** and **save_forest** can call many of the same routines, **save_forest** uses the **file_options** flag bits to determine whether to call the listed routines. **Create_model_files** calls all the routines listed. The flow chart illustrating **run_model** is shown in Figure 22.

Framer	Level 1	Level 2	Level 3
RUN			
	run_model		
		create_model_files	
			write_lines_of_notes
			write_gen
			write_stand
			write_one_species
			write_STEMS
			write_mgmt
			write_weather
			write_defol
			write_ISTR
		save_forest	
			write_gen
			write_stand
			write_one_species
			write_STEMS
			write_mgmt
			write_weather
			write_defol
			write_ISTR
			write_notes
		Stand (FORTRAN subroutine)	
		screen_background	
		erase_model_files	

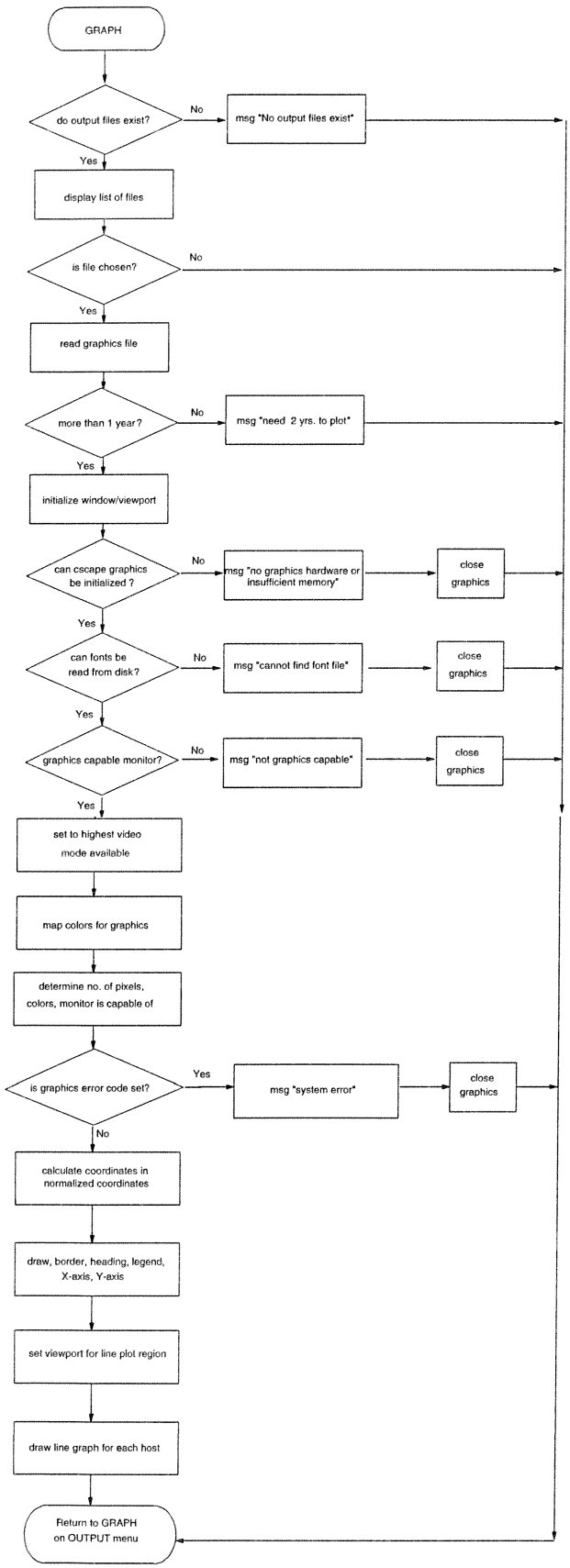


Figure 21. Graph output files.

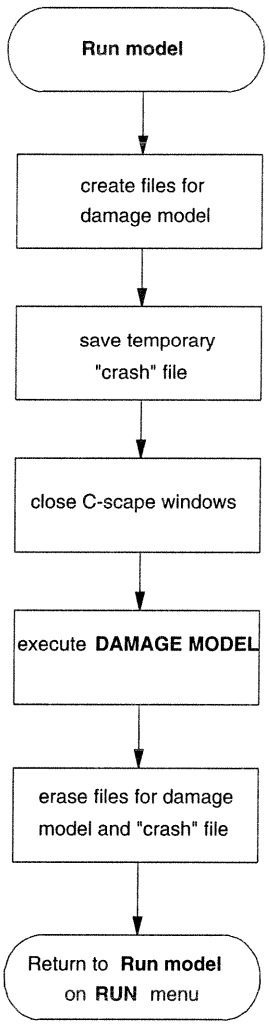


Figure 22. Run model.

Quit

Upon exiting the program, the user is given a chance to save edited data and generated output files. **EXIT** is a boolean variable used in a "while" loop in the main function call to **framer_menu**, the controlling menu for the software. While **EXIT** is set to FALSE, the framer menu is displayed.

When Quit is selected from the framer menu, **exit_routine** is called. **EXIT** is set to FALSE. If **EDITS_MADE** is TRUE, the user will be given the option of saving the file before exiting.

The **EDITS_MADE** flag indicates whether editing has been done during the current session. If the user decides not to save the file, **EXIT** is set to TRUE and the program terminates.

If the user decides to save the file, **save_forest** is called to save the file, **EXIT** is set to TRUE, a call is made to **output_delete_on_exit**, and the program terminates.

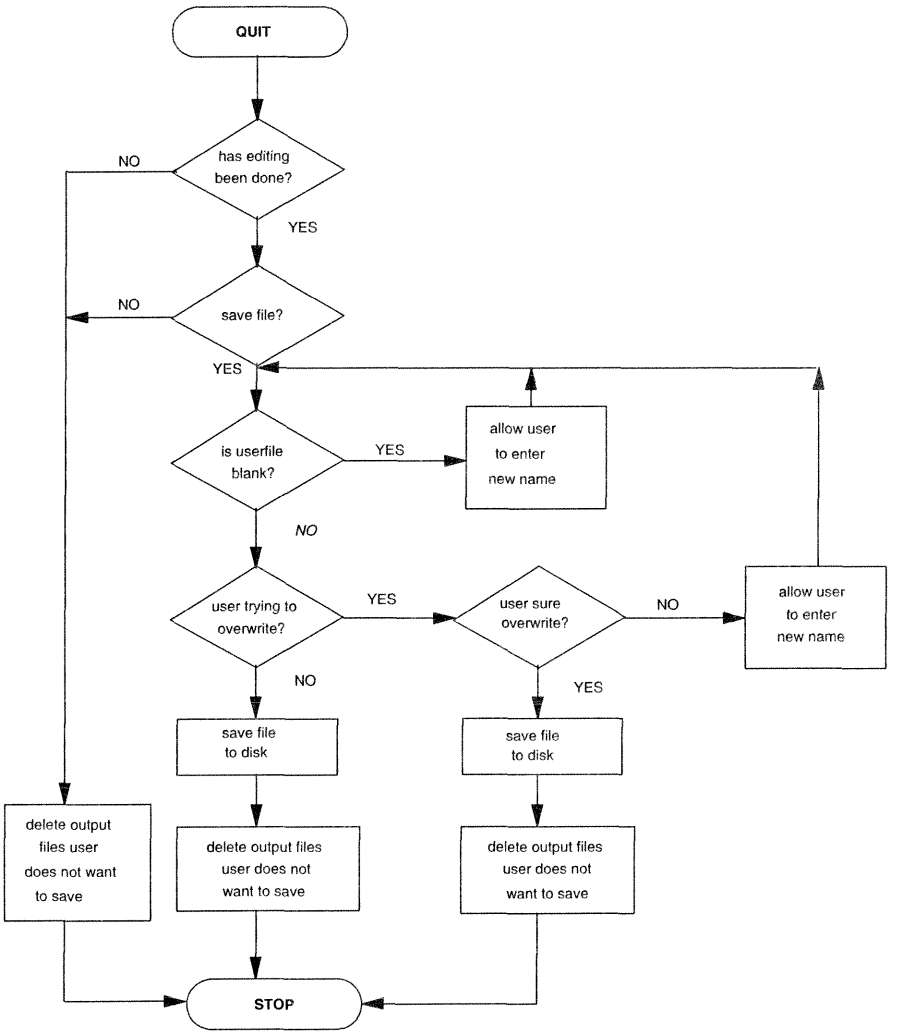


Figure 23. Quit.

Framer	Level1	Level 2	Level 3
QUIT			
	exit_routine		
		verify_choice	
		forestfile_edit	
		save_forest	
			write_gen
			write_stand
			write_one_species
			write_STEMS
			write_mgmt
			write_weather
			write_defol
			write_ISTR
			write_notes
		output_delete_on_exit	
			output_to_save

Output_delete_on_exit ensures that the user does not exit without having a chance to save generated output data. If output files exist, **output_to_save** is called to allow the user to save output files. **Output_to_save** is described in the framer-menu OUTPUT section. The flow chart illustrating **exit_routine** is shown in Figure 23.

Non-Framer Modules

Three software modules fall outside of the framer menu hierarchy and are discussed in this section.

Main

The function **main** loads the C-scape interface, displays introductions to the model, loads default stand model parameters, enables the mouse, and places headings and menus on the screen.

Main first calls initialize. **Initialize** sets the display adaptor to text mode using the C-scape function **disp_Init**. **Init_and_show_pcx** displays a graphics file in a pcx file format. If the software is running in monochrome mode, **Mono_display_check** translates the colors to monochrome. The function **intro** displays a short program introduction.

Level2	Level3	Level4
initialize		
	init_and_show_pcx	
		display_pcx
	mono_display_check	
	screen_background	
	intro	
		initialize_help
	initialize_output	
	read_forest	
		read_gen
		read_stand
		read_one_species
		read_STEMS
		read_mgmt
		read_weather
		read_defol
		read_ISTR
		read_notes
frame_menu		

Initialize loads the mouse (if a mouse driver is present) and then calls **screen_background** to paint the screen background on color monitors. The linked-list pointers are set to NULL, the weather array is initialized, and **initialize_output** is called to initialize the output structures. **Read_forest** is called to read default values from the "DEFAULT" disk file and assign them to the stand-model variables. The **file_options** parameter is the first parameter read from the file. Since the flag bits are set in the default value of **file_options**, all default file parameters are read in. **File_options** is then reset to zero so only essential data will be saved if no further editing is done.

When control returns to **main**, **frame_menu** is called within a "while" loop so that the framer menu is displayed so long as the boolean variable **EXIT** is FALSE. The program terminates after the framer menu is exited. The flow chart that illustrates **main** is shown in Figure 24.

Data Access

When a user tries to save a file after data has been edited, only specific data will be saved depending on the type of data that have been edited (stand, weather, stand management, or general). Thus, the program must keep track of what was edited and save data accordingly. The bits for the integer variable (**file_options**) are used as flags to indicate whether general, stand, stand management, or weather data should be contained in the user's data file. **File_options** individual bits are used to

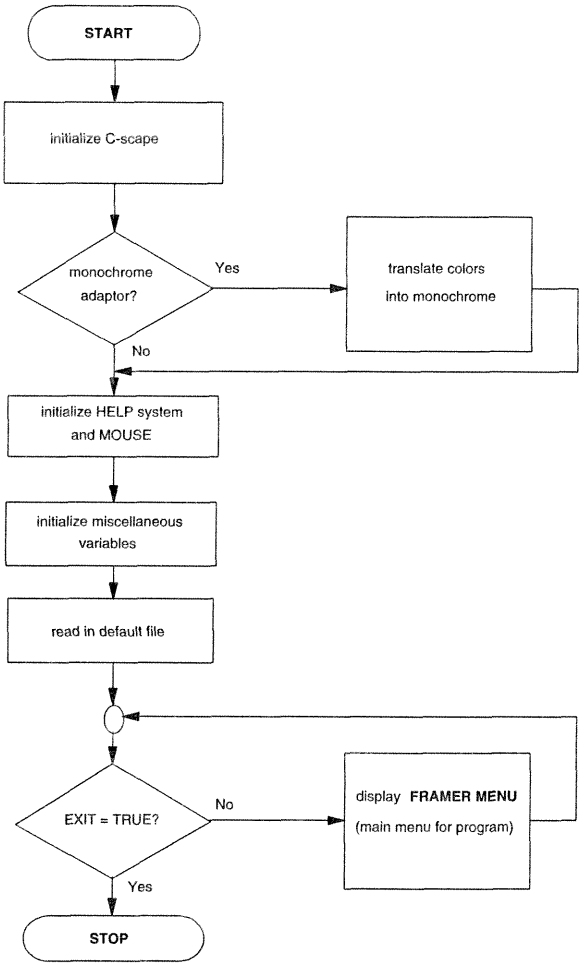


Figure 24. Main.

store YES/NO values. The values of the flag bits for **file_options** are:

General	0001	0x01
Stand	0010	0x02
Weather	0100	0x04
Management	1000	0x08

File_options is the first variable written to the user's forest file. When an existing file is accessed, **file_options** bits are checked to see what data should be read into the program. After the default forest file is loaded upon entry into the program, **file_options** is set to 0 (0000). When the user enters a section to edit data, **file_options** is changed accordingly. The code consists of two routines. The first is named **save_forest** and the second is named **read_forest**.

Save_forest opens the forest file and writes **file_options** and **NHOST** to the file. **Save_forest** checks the **file_options** flag bits (general, stand, weather, and stand management), and if the appropriate bit is set, the corresponding routine is called. The flow chart that illustrates **save_forest** is shown in Figure 25. **Read_forest** checks the **file_options** flag bits and if the appropriate bit is set, the corresponding routine is called.

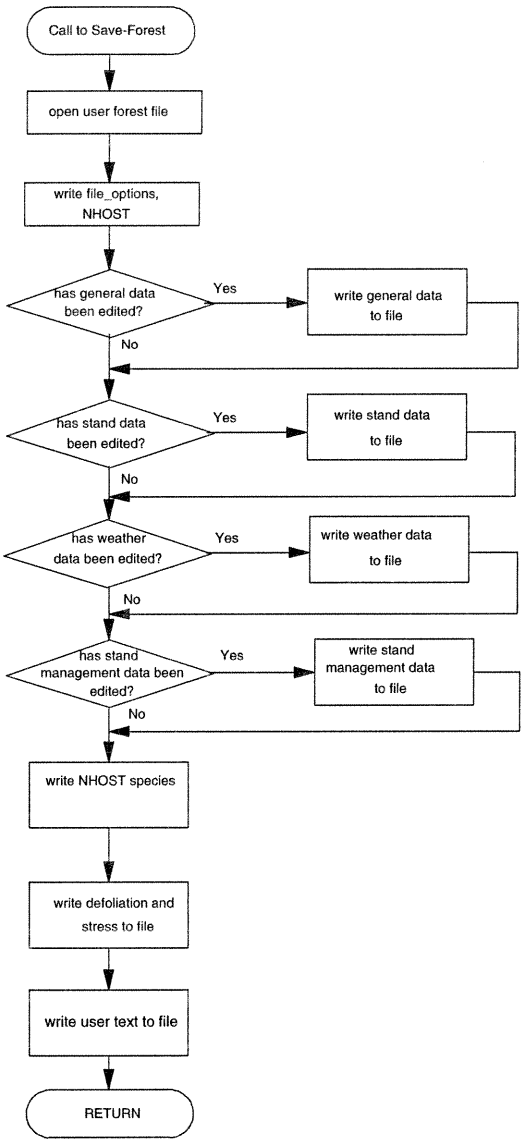


Figure 25 Save forest.

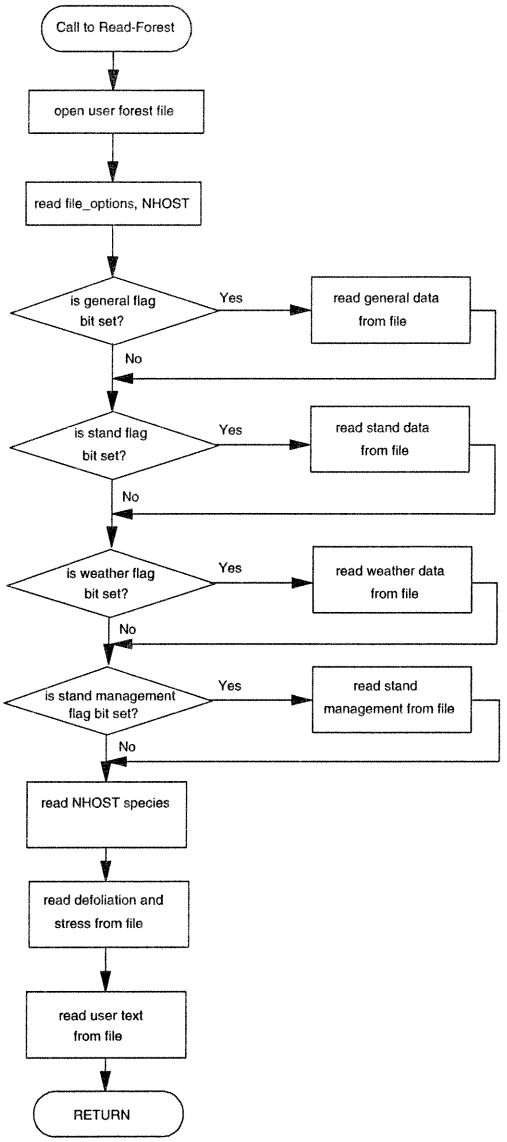


Figure 26. Read forest.

The flow chart that illustrates **read_forest** is shown in Figure 26.

Help System

The C-scape help system is an indexing engine that retrieves messages from a help file. The help file is an ASCII text file that contains the text of the help messages and the help index. C-scape displays a help message when F1 is pressed. The help system is invoked by calling the **help_Init** function during program initialization. The help system format closely parallels the code structure. When the screen order of the variables is changed, the help file order must be changed as well.

Default Data and Transfer Files

The interface must access two data files when it begins: `DEFAULT` and `SPECIES`. When the user saves a data file, the file name is <username>.`INP` (any name can be supplied by the user but the extension will always be `.INP`.) There are five files created by the interface for use by the model: `GENDATA`, `TREEDATA`, `ANNDDSUM`, `DFOLDATA`, `STRESYRS`, and `USERNOTE`. A temporary file, `CRASHED.INP`, is created in case the program aborts during model execution. `CRASHED.INP` and the other temporary files are deleted when the simulation properly returns control to the interface.

There are up to six output files created by the model which can be displayed and saved by the interface: `STANDTBL.D1S`, `GDBH.D1S`, `GVOLUME.D1S`, `DBUGDAMG.D1S`, `GSTEMNUM.D1S`, and `GBASALAR.D1S`. The simulation is described within the `STANDTBL.D1S` tabular output file. This file contains input parameters, initial conditions, and annual data summaries. `DBUGDAMG.D1S` contains details for tracking intermediate calculations. The

other output files are ASCII-text files that can be graphed within the interface. GRAFHOST.D1S is generated by the model for use by the interface graphics routines.

Default data are stored in the disk file named DEFAULT. This file is retrieved from the current directory when the program begins. DEFAULT is an ASCII-text file separated into six sections or groups of records. The first section contains header information, followed by general simulation data, stand-site data, tree species-specific data, weather data, and defoliation data.

Besides the six sections of default data there are two additional data sets that may be needed when the model is run: one when management prescriptions are to be simulated and a second when the user chooses to prescribe specific years for additional tree stress mortality beyond the nominal background and that from defoliation. If the user chooses to exercise the Management Option, one data block is created for each stand-management prescription specified. The blocks are written together into the Saved file that the user creates under the File Option menu item. They are also added to the end of the stand-tree data file that is passed to the simulator. The notes describing the simulation are contained in the final block of data in either file.

The DEFAULT file is described first. This file contains several data items or parameter values that are applicable only to the linked version of the stand and gypsy moth models within the full Gypsy Moth Life System Model (GMLSM). These data remain in the DEFAULT file and are passed to the stand-damage simulator for compatibility with the full GMLSM, but they are not accessible for editing and do not affect the simulation of the stand-alone version of the model. Retaining them in this file and the associated code minimizes the number of distinct file variants required for this modeling system. Header information resides in the first line of the DEFAULT, CRASHED.INP, or the user-saved input file <username>.INP:

Table 1.--Description of header information records

Line no.	Col. nos.	Data type[a]	Name[b]	Description	Range[c] (default)
1	1	int	file_options	Data access flag using bits 0-3.	0≤i≤15
	3	I1	NHOSTS	Number of host-species data blocks.	0<x≤6,(3)
	5	I1	screen_color	Monitor type.	0,1

[a]Data types are described in a FORTRAN format. When variables are local to the C interface, the C convention is used. mTn := format specification where m is a repeat factor, T specifies the format, and n is field length. For example, 3F4.2 means that three real numbers are to contain four digits, the last two are to the right of the implied decimal place. Format specifiers are: A = character or string; E = real mantissa-exponent notation; F = real fixed notation; I = integer; L = logical (T or F); X = space or blank; / = move to next record (line) for more input.

[b]Variable names are identical to those in the original FORTRAN code except for those local to the interface, which use the C naming conventions.

[c]When a range is inappropriate, an example is provided.

General data is passed to the stand simulator as a 13-record file. This block always contains a single line with two data, the number of tree species to be simulated (**NHOSTS**), an integer in columns 2 and 3, followed by the number of lines of notes prepared by the user for inclusion in the output file header (**NLINES**), an integer in columns 5 and 6. The second line of the GENDATA file contains a 12-character field that includes the name of the input data file used to save the data that will generate the simulation. Note that if input data have not been saved on disk, "default" will remain in this field. This line is followed by the next 11 lines (or records) of the DEFAULT file in the format:

Table 2.--Description of general data records

Line no.	Col. nos.	Data type	Name	Description	Range (default)
2	1-20	A20	name	User supplied Name.	'name'
3	1-20	"	job	User supplied Job title.	'job'
4	1-20	"	site	User supplied Site name.	'site'
5	1-20	"	date	User supplied Date.	'date'
6	2	L1	STSUBM	True if stand model is to run.	(T)/F
	4	"	GMSUBM	True if running the gypsy moth submodel.	(F)/T
6	6	L1	DEBUG	True if debug output is desired.	(F)/T
7	2-4	I3	NYEARS	No. of years to simulate.	i≥1
	6-9	I4	ISYEAR	Calendar year to label start of simulation and variable to increment current year through the simulation.	1979
8	2-3	12	IPYEAR	Output interval for stand table. "0" indicates to use IVIEW on lines 8-9.	i≥0
	5	L1	STABLE	True requests stand table output.	(T)/F
	7-8	I2	IGYEAR	Interval (years) for printing output to nontable output files.	(10)
	10	L1	TGSTEM	True if stem count file is to be written out for external use.	(F)/T
	12	L1	TGBA	True for basal area file.	(F)/T
	14		TGVOL	True for volume file.	(F)/T
	16	L1	TGDBH	True for diameter file.	(F)/T
9	2-5	I4	IVIEW	Specific years that stand tables are	
10	7-10		(I)	to be printed in the output; these	
	12-15			are used only if IPYEAR = 0.	
	17-20				
	47-50				
11	1-7	6I7	ISEED	Random number generator seeds, for:	2653
	8-14		(I)	1. establish new trees; 2. not used;	5107 NU[a]
	15-21			3. calc. stress mortality of trees;	3745
	22-28			4. gypsy moth winter temp. induced	5492 NU
	29-35			mortality; 5. generate weather data;	8729 NU
	36-42			6. first instar gypsy moth dispersal.	1735 NU
				Only the first and third are used by the model and available for editing.	
12	2-6	F5.2	TFAC	Temperature scaling factor, 1.0 means no change, 0.9 means a 10% reduction.	0.0≤x10.0
	8-10	I2	NWEAYR	No. of years of weather data.	i≥1 NU
	12-17	F6.2	TRETHR	lower threshold for tree degree-day accum.	(42.0) NU
12	19	I1	IWFORM	format choice for reading weather data, three choices for IWFORM =: 1 - (4(/,10F6.1)); 2 - (2(/,16F3.0)); anything else, use (2(/,16F5.1)). These are only used with the gypsy moth model and are not used by the stand	1,(2),3 NU

Continued

Table 2.--Continued.

Line no.	Col. nos.	Data type	Name	Description	Range (default)
				model nor are they available for editing.	
	21	I1	IWOPT	Weather Options: IWOPT =: 1 - one year of daily data; 2 - more than one year of daily data; 3 - generate daily weather data from monthly averages; 4- use annual degree-days. Only annual degree-day data is used for the stand model.	(4) NU 1,2,3
	23	L1	TEMPC	True if temperatures are in celsius.	(F)/T NU
	25	"	LRAIN	True if user wants to include rain in the simulation.	(F)/T NU

[a]This datum is not used (NU) by the Stand Damage Model in its stand-alone configuration; it will not be accessible to the user to edit.

As mentioned earlier, the number of lines of notes (**NLINES**) the user has written to the notepad within the Setup Window is passed in the first record of GENDATA. If **NLINES** is greater than zero, the interface writes the notes to file USERNOTE for display in the output summary table file.

Stand data contains only data for the stand as a whole. Stand data, tree-species data, and stand management data are passed to the model through the TREEDATA file.

Table 3.--Description of stand data records

Line no.	Col. nos.	Data type	Name	Description	Range (default)
13[a]					
14	2	I1	ISITE	Site moisture index: 1 = wet, 2 = medium, 3 = dry.	(1),2,3
	4	L1	METRIC	True if metric units used.	(F)/T NU
	6	L1	LDEFOL	True if defoliation data are to be read from file DEFOLDAT.	(T)/F
15	2-7	F6.2	PLOTAR	Total area of plots used to estimate stocking for full stand.	x≥0.01 (1.0)
	9-14	F6.2	STNDAR	Stand area being simulated.	x≥0.01 (1.0)
	16	I1	IBOUND	Overstory/Understory boundary type:1/not 1 If 1 then height used, else diameter.	(2),1
	18-24	F7.1	HOVER	Boundary height in cm or ft for tree height (IBOUND=1), or boundary diameter in cm or inches for diameter at breast height (DBH) (IBOUND = 2).	x>0.0 (5.0)
16[a]					
17	2-5	F4.1	TLIGHT	annual insolation factor, eqn. 10[b].	(1.0)
	7-13	F7.5	TKL	K in eqn. 10[b].	(0.0002)
18	2-7 9-14	F6.2 (I,J)	SHX	I=1-6, J=1-4: parameters for effect of light on diameter growth; x coordinates for Fig. 4[b].	
19		6(1X,F6.2)/		shade-tolerance class, I, varies	
20		"		across line or row, while	
21		"		x coordinate, J, varies down columns.	
22	2-7	6(1X,F6.2)/	SHY	I=1-6, J=1-4: parameters for effect	

Continued

Table 3.--Continued.

Line no.	Col. nos.	Data type	Name	Description	Range (default)
	9-14		(I,J)	of light on diameter growth; y coordinates for Fig. 4[b].	
23		6(1X,F6.2)/		shade tolerance class, I, varies	
24		"		across line or row, while	
25		"		y coordinate J, varies down columns.	
26	2-6	F5.2	STRFAC	1: Nonstress mortality factor; 2: Stress-	(0.00) 0.0≤x≤1.0
	8-12	F5.2	(I)	induced mortality factor, see SDIE, eqn.19[b].	(0.15) 0.0≤x≤1.0
	14-20	F7.4	PLOTSZ	Size of the area to be used in calculating the influence of surrounding trees on shading of a given tree.	(0.0833)
27	3	I1	ISOPT	Stress option, 1 for random stress, 2 for designated years (see ISTR(I)).	0,(1),2
	3-6	F4.1	STRESS	Probability that additional stress	(0.0)
	7-10	F4.1	(I)	will occur by site index, I = 1-3. Note:	(0.0)
	11-14	F4.1		variable ISITE determines which is active.	(0.0)
28	1-7	F7.4	RSMULT	Effect of relative stocking on tree crowding index, see eqn. 12[b].	(0.0025)
29	1-4	6(1X,F4.0)	STOKX	I=ISHADE(IHOST,2), shade tolerance class	
30	5-8		(I,J)	for recruitment; I = 1,...,6.	
31	9-12		(6,4)	J=interpolation points, also see STOKY	
32	13-16			x coord. for shade tolerance effect on recruitment; J = 1,...,4.	
33	1-5	6(1X,F5.2)	STOKY	I=ISHADE(IHOST,2), shade tolerance class	
34	6-10		(I,J)	for recruitment; I = 1,...,6.	
35	11-15			J=interpolation points, also see STOKX	
36				y coord. for shade tolerance effect on recruitment, see Fig. 9[b] and Table 2[b].	
37	2-11	F10.7	STOCKS	J=1,2,3;I=1: stocking class parameters,	
	13-22		(I,J)	3 per quadratic, 3 stocking classes,	
	24-33			species specific (in ISTOKG(IH))	
38-39				I = 2, 3 for STOCKS; similar to L-37.	
40	2-5	F4.2	DFOLDE	Slope of defoliation effect curve; and the maximum proportional decrease in this year's foliage production (at 100% defoliation) due to last year's defoliation (DEFL(IH,IS)) in TREE1.	(0.15) 0.0≤x≤1.0
	7-10	F4.2	CRWDMN	Minimum value for CROWD in eqn. 12[b]. See TREE2.	(0.75) 0.0≤x≤1.0
	12-15	F4.2	SHADMN	Minimum value for SHADE in eqn. 16[b]. See TREE2.	(0.05) 0.0≤x≤1.0
	17-22	F6.3	SGMORT	Effect years of slow growth (NSLOW) will have on tree mortality rate. Exponential rate coefficient for calculating GDIE, eqn. 18[b].	(-0.084) -0.0≤x≤0.0
41	2-11	F10.8	DFLC	J=1,2;I=1,2: Parameters for eqn. 13[b]	
	13-22		(I,J)	(light) and eqn. 14[b] (heavy) for	
	24-33			predicting DEFIND (eqn. 16[b]), the effect	
	35-44			of current defoliation on diameter growth.	
42	2-4	I3	NSTRYR	Number of years that stress is to be included at prescribed years set by user; used only when ISOPT=2; only limit is the length of simulation (max. = 100 years).	(0) 0≤i≤100
	6-11	F6.2	DLENBASE	Base diameter class width. All classes are same width; they start at 0.0; for int. n,	(2.0) 2.0≤x≤4.0

Continued

Table 3.--Continued.

Line no.	Col. nos.	Data type	Name	Description	Range (default)
				class midpoints are n*DLENBASE + DLENBASE/2.0.	
	13	I1	ICAT	Output category: 1 = overstory, 2 = understory, 3 = both.	(1),2,3
43	2-9	F8.6	EPSIGR	Parameter to allow bypass of diameter growth when stem count is very small for a cell. See subroutine TREE2.FOR[b].	(0.0) 0.0≤x≤0.01
	11-18	F8.6	EPSITR	Parameter to allow bypass of transfer between diameter class cells when stem count is very small. See subroutine TREE2.FOR[b].	(0.0) 0.0≤x≤0.01
	20-27	3F8.6	EPSIMG(I)	Parameters for controlling inclusion of small	(0.0)
	29-36			stem counts in tree removal calculations; I =	(0.0)
	38-45			1, 2, or 3. See FORTRAN documentation for more detail; also see subroutine TREE2.FOR[b].	(0.0)
	47-54	F8.6	PRTMIN	Parameter to suppress inclusion of small stem counts in output table.	(0.0)

[a]This is either a blank line or is for program comments;
[b]Refers to companion document: "Description of the Stand-Damage Model: part of the Gypsy Moth Life System Model" (General Technical Report NE-208).

Tree data contains one set of records or lines of input for each tree species in the stand. Thus, it is of variable length and formatted in blocks. The description of the tree data file follows.

Table 4.--Description of tree data records

Line no.	Col. nos.	Data type	Name	Description	Range (default)
44	1-2	A2	TNAME (IH)	Species code of the first of the default tree data groups, each species takes eight lines.	
45	2-3	I2	ISHADE (IH,1)	Recruitment shade tolerance. index, see Table 2[a] for values.	1≤x≤6
	5-6	I2	ISHADE (IH,2)	Diameter growth shade tolerance index, see Table 2[a] for values.	1≤x≤6
	8-15	F8.4	RESTIN (IH,1)	Max no. larvae per 100 cm^2 on lower boles, Table 2[a].	x≥0.0
	17-24	F8.4	RESTIN (IH,2)	Max no. larvae per 100 cm^2 on upper boles, Table 2[a].	x≥0.0
	26-32	F7.1	GROWR	Base growth rate for eqn. 15[a].	
	34-39	F6.0	COLD	Min DD accum expected within range of this species.	x≥0.0
	41-46	F6.0	HOT	Max DD accum expected within range of this species.	x≥COLD
	48-53	F6.3	SLOWD	Minimal diameter growth to avoid additional mortality.	x≥0.0
46	2-7	F6.2	CR1	coef. C1 for eqn. 6[a], see Table 5[a].	NU
	8-14	F7.4	CR2	coef. C2 for eqn. 6[a], see Table 5[a].	NU
	16-21	F6.2	CR3	coef. C3 for eqn. 6[a], see Table 5[a].	NU
	23-28	F6.3	CR4	coef. C4 for eqn. 6[a], see Table 5[a].	NU
47	2-8	F7.1	DMAX	Max diameter for eqn. 15[a], see Table 3[a].	
	10-16	F7.0	HMAX	Max height for eqn. 15[a], see Table 3[a].	
	18-21	F4.0	AGEMAX	Max age for eqn. 17[a], see Table 3[a].	
	23-30	F8.5	F1	foliage param. for eqn. 5[a], see Table 2[a].	
	32-39	F8.5	F2	foliage param. for eqn. 5[a], see Table 2[a].	
	41-46	F6.1	SURFAR	Biomass to surface area conversion factor see Table 4[a].	
	48-49	I2	ISTOKG	Relative stocking index, Table 2[a].	
	51-56	F6.0	RECRUT	Max no. stems recruited/yr, Table 4[a].	
48	2-5 7-10 ...	9(1X,F4.2)	FDIE	(IH,J,K)J=1,2,3;K=1,2,3; proportion of trees growing on site K that die following J consecutive years of defoliation > 65%.	
49	2-54 ...	4(1X,I4)	IDHIST	(IH,J,K) J=1,2,3;K=1,2; percent defoliation by strata J = 1:upper, 2:lower, 3:shrub/ground and prior year: K = 1:last yr, K=2:2 yr old.	(0.0)
50	2-7 9-14 etc.	(1X,F6.1)	STEMS	(IH,ID) Stem count by diameter class, the program assumes that ALL stems in the stand are included, ID = 1-10.	
51	2-7	10(1X,F6.1)		continue, for ID = 11-20 etc.	

[a]Refers to companion document: "Description of the Stand-Damage Model: part of the Gypsy Moth Life System Model" (General Technical Report NE-208).

Lines 44-51 are repeated once for each of the three default species. Three default species end with line 67 of the DEFAULT file. Each tree species has its own set of defaults, so none are shown in the table. See the tables referenced in the "Description of the Stand-Damage Model: part of the Gypsy Moth Life System Model" (General Technical Report NE-208).

The SPECIES data file contains 20 blocks of six records each and are precisely parallel to lines 49-54. This file provides species parameters and default values. Each stem count (similar to records 55-56) is initialized to zero (0.0). However, a single stem is placed in the smallest diameter class to assure an initial population. See Tables 1-5, 7, and 8 in the "Description of the Stand-Damage Model: part of the Gypsy Moth Life System Model" (General Technical Report NE-208).

Default weather and defoliation data follow. A variable representing annual accumulated degree-days corresponds with each simulation year. These data are written to file ANNDDSUM.

Defoliation data are supplied for 3 years. Following an unused note line, the number of years of defoliation to be simulated is provided, followed by three lines (records) for each year of defoliation.

Table 5.--Description of weather data records

Line no.	Col. nos.	Data type	Name	Description	Range (default)
68-72	2-7	F6.1	ANNDD(I)	Annual accumulated degree-days above 42 degree F. starting from the first day of the year. I from 1 to 5 since the default is a 5-year simulation.	(4570.0)

Table 6.--Description of defoliation data records

Line no.	Col. nos.	Data type	Name	Description	Range (default)
73[a]					
74	1-2	int.	number_of_ defol_yrs	Number of defoliation data structures.	(3)
75[b]	2-5	I4	IDEFOLY	Year of next defoliation event.	(1979)
76	2-4	6(1X,F3.0)	DEFL(IH,1)	Overstory defoliation for each tree species.	(0.0)
	6-8				...
	...				0.0≤x≤100.0
77	2-4	"	DEFL(IH,2)	Understory defoliation for each tree species.	(0.0)

[a]This is either a blank line or is for program comments;
[b]Lines 75-77 are repeated twice for the years 1980 and 1981.

Management data is another group of data generated by the interface that does not have default data associated with it. The management of the stand is simulated by implementing one of the management options within the interface. In the default situation, only the first of the two lines that follow are written to the TREEDATA file. When management is selected, the variable **MTOTAL** is reset from its default value of zero to the number of management years scheduled. The management information is written to the TREEDATA file as the final block of records in that file.

Stress mortality information may be prescribed by the user. This information is represented by specific years in which stress mortality is beyond that directly induced by the defoliation history. The file STRESYRS contains the stress years selected. This file has one record for each calendar year (**ISTRYR**) with the date in columns 2-5.

Table 7.--Description of management data records

Line no.	Col. nos.	Data type	Name	Description	Range (default)
n		--			
n+1	1-2	I2	MTOTAL	Number of years of requested management.	(0) 0≤i≤99
n+2	2-5	I4	MYEARS	Calendar year for a management action.	
	7	I6	MANAGE	Management type: 1 = proportion of stems; 2 = target residual stem count.	
n+3[a]	2-7	F6.1	CUTMIN (IH)	Minimum diameter (cm) of species IH to be removed.	(15.0)
	9-14	F6.1	CUTMAX (IH)	Maximum diameter (cm) of species IH to be removed.	(200.0)
	16-21	F6.2	PCUT/ TARGET (IH)	Depending on the value of MANAGE, this is either the proportion to cut, PCUT, or the residual per acre stem count, TARGET, following the removal.	(0.50) 0.0≤x≤1.0 (50.0) x≥0.0

[a]Line n+3 is repeated for each tree species present in the stand to complete one management implementation block. The blocks are repeated for each management year.

Installation Software

The installation software for the Stand-Damage Model is written in the language C for DOS microcomputers. The software allows the user to choose an individual path and directory for model installation, decompresses zipped model files, and installs the model files to a designated path.

Main calls the routines shown in Level 1. **Main** first calls **welcome** to display a brief introduction to the installation software. A beep is sounded if anything other than a carriage return (CR) or the Escape key is entered. If a carriage return is entered in **welcome**, **get_install_drive** is called. It prompts the user for the installation drive being used. A case statement is used to distinguish between the different drive possibilities (A-Z).

Level 1	Level 2	Level 3	Level 4
welcome			
	getkey_stroke		
	exit_program		
	get_current_directory		
get_install_drive			
	install_drive_existence		
		get_install_drive	
	get_install_drive		
get_target_drive			
	target_drive_existence		
		check_drive_storage	
			get_target_drive
		get_target_drive	
	get_target_drive		
get_base			
	existing_path		
		existing_path	
	get_base		
get_directory			
	new_directory		
		new_directory	
	get_directory		
get_monitor_type			
insure_path			
	insure_path		
unzip_damage			
reset_starting_path			
is_damage_installed			
installation_complete			
installation_not_done			

Install_drive_existence is passed the drive letter chosen. If the drive does not exist, the computer beeps and **get_install_drive** is called recursively.

Several functions and techniques are used throughout. **Getkey_stroke** retrieves a single character from the user without echoing or waiting for an entire line of input. **Exit_program** is called when the Escape key is pressed. **Exit_program** resets the starting path and exits the program.

Case statements are used to handle user entries. When an invalid entry is made within function X, the default case statement makes a recursive call to function X. Circular function calls is another commonly used technique. Say that function X calls function Y. If there is an error in function Y (e.g., the designated directory does not exist), function X is called from function Y in a circular manner to allow the user to attempt to make a valid entry.

After a valid installation drive has been obtained, **get_target_drive** is called to obtain the drive in which the model is to be installed. **Target_drive_existence** is passed the letter chosen. If the drive does not exist, a recursive call is made to **get_target_drive**. **Check_drive_storage** is called to ensure that there is enough free disk space to accommodate the model. If disk space is inadequate, a warning is displayed and **get_target_drive** is called recursively.

Get_base is called to obtain the directory base where the installation should take place. The user can choose the target drive as the base or can enter an alternative directory base. If an alternative base is chosen, **existing_path** is called and the user can enter an existing path where the model should be installed. If the entered path is not valid, **existing_path** is called recursively.

After **get_base** is exited, **get_directory** is called to allow the user to designate the directory (at the end of the path) that will contain the stand files. The user is given three options: 1. Choose to install the model in the existing directory displayed. Nothing needs to be done in this case since **target_full_path** already contains the path. 2. Choose to establish an alternate name for the directory. **New_directory** is called and the user enters a new directory. If the directory name entered is over eight characters long, **new_directory** is called recursively. Otherwise, the new directory name is created. 3. Choose to place the files in a directory named "DAMAGE." **Target_default_path** ("DAMAGE") is concatenated with **target_full_path**.

Get_monitor_type is then called to determine whether the user wishes to use a grey-scale VGA monochrome monitor. The default situation is to choose color or monochrome. Note that the stand-damage software can detect whether the monitor is color or monochrome and makes appropriate color adjustments. However, grey-scale VGA monitors cannot be detected and must be explicitly called for.

Insure_path is called next. It ensures that both the installation drive and the target drive and path are what the user wants. The user presses the carriage return to use the configuration shown or presses Escape to abort the installation.

The model files are compressed to fit on 360K floppy disks. **Unzip_damage** decompresses these files. A system call unzips the compressed files from the installation drive to the targeted path. If the user has chosen to install the model on a computer with a grey-scale VGA monitor, a file named DEFAULT.MON is copied to DEFAULT. DEFAULT.MON is then erased.

Reset_starting_path is called to change to the drive and directory that were active when the installation program began. **Is_damage_installed** is called to ensure that all the model files are present in the designated directory. If any files are missing, or if the system call made in **unzip_damage** was unsuccessful, **installation_not_done** is called to display an error. If the files are present and the system call was successful, **installation_complete** is called to inform the user that the installation was successful.

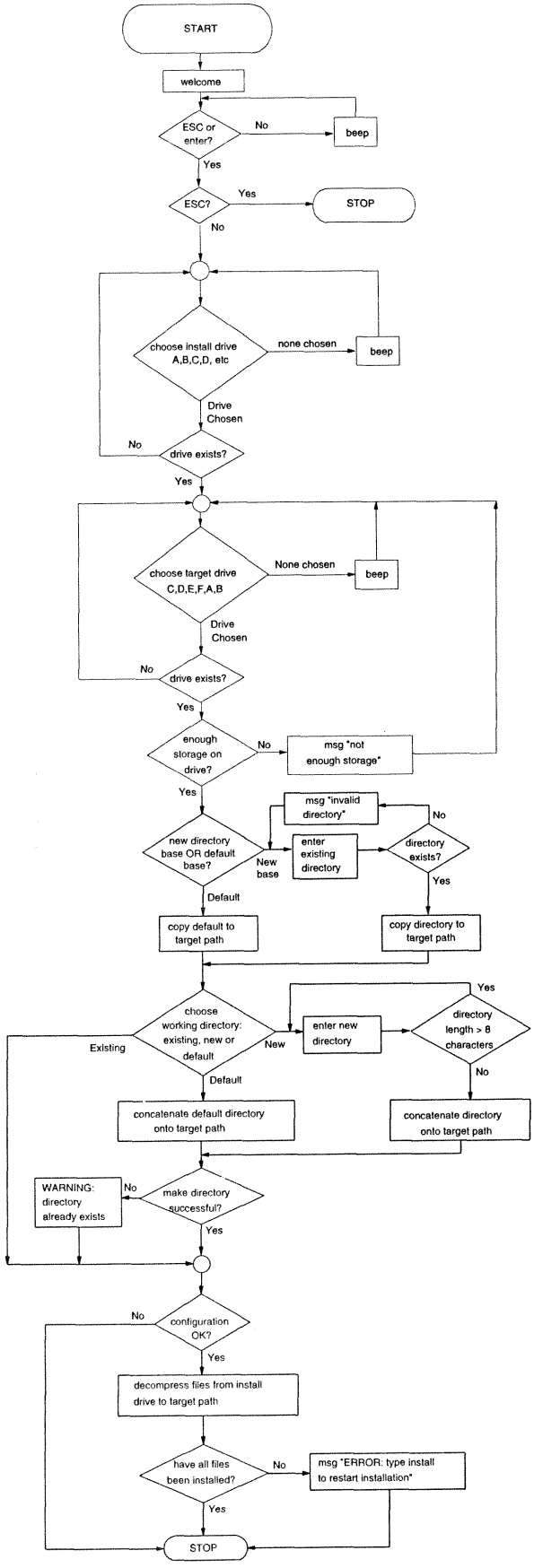


Figure 27. Installation software.

Figure 27 is a flow chart that illustrates the installation program.

Acknowledgments

We thank Kurt Gottschalk, Andrew Liebhold, and Mark Twery, USDA Forest Service, Alexei Sharov, Virginia Polytechnic and State University, and Xu Rumei, Beijing Normal University, for reviewing the model interface. We also thank David Feicht and Sandra Fosbroke, USDA Forest Service, for developing data sets to be used for the model. Also, thanks to all our beta testers for providing suggestions for improvements.

Literature Cited

Colbert, J. J. and Sheehan, Katharine A. 1995. **Description of the Stand-Damage Model: part of the Gypsy Moth Life Systems Model.** Gen. Tech. Rep. NE-208. Radnor, PA: U.S. Department of Agriculture, Forest Service, Northeastern Forest Experiment Station. 111 p.

Colbert, J. J. and Racin, George. 1995. **User's guide to the Stand-Damage Model: a component of the Gypsy Moth Life Systems Model.** Gen Tech. Rep. NE-207. Radnor, PA: U.S. Department of Agriculture, Forest Service, Northeastern Forest Experiment Station. 38 p.

Microsoft Corporation. 1990. **C 6.0 manual.** Redmond, WA: Microsoft Corporation.

Oakland Group, Inc. 1989. **C-scape manual.** Cambridge, MA: Oakland Group, Inc.

Source Code

```
/*************** STAND-DAMAGE MODEL INTERFACE DEFINEs ****************/
#define ROWS_FOR_ONE_TREE     6  /* rows for one host species in species file */
#define MAX_HOSTS            12  /* max number of host species */
#define MAX_SIM_YRS         100  /* max # of years for simulation */
#define MAX_MGMT_YRS         15  /* max # of list elements for management */
#define MAX_DEFOL_YRS       100  /* max # of list elements for defoliation*/
#define MAX_VIEWS            20  /* max # of IVIEW */
#define MAX_INP_LIST         15  /* max # of input files (*.inp) displayed */
#define MAX_OUTPUT_FILES      6  /* max # of output tables */
#define MAX_NOTE_SIZE       500  /* max # of bytes in user notes */
#define NUM_VALID_CHARS      68  /* # of valid characters for file names */
#define LINE_SIZE            73  /* max # of characters in 1 line of user notes*/
#define OUTPUT_BYTES      32500  /* bytes used to display output file*/
#define INTRO_BYTES        3000  /* bytes used to display intro file */
#define HELP_BYTES         2500  /* bytes used to display single help message */
#define POSIBL_SPECIES       22  /* # of possible host species */
#define TARGETED              2
#define beep()                hard_Speaker (1500, 10)
#define STABLE                0  /* these next 6 refer to table output */
#define TGSTEM                1
#define TGBA                  2
#define TGVOL                 3
#define TGDBH                 4
#define DEBUG                 5
#define DELAY                50 /* hard_Pause time delay; 0.5 seconds */
#define PRIVATE                  OSTATIC
#define SUCCESS               0
#define SI                    0
#define ENGLISH               1

/* file_options flag bits - see file_options */
#define  GEN          0x01
#define  STAND        0x02
#define  WEA          0x04
#define  STDMAN       0x08

/* function keys  */
#ifndef FN1
#    define FN1            0x3B00
#endif

#ifndef FN2
#    define FN2            0x3C00
#endif

/* for linked list */
#define SUCCESSFUL_DELETION  0
#define KEY_NOT_FOUND        1
#define LIST_EMPTY           2
#define DUPLICATE_KEY        4
#define NO_DUPLICATE_KEY     5
#define NULL_MGMT_PTR        (MGMT_YRS *)NULL
#define NULL_DEFOL_PTR       (DEFOL_YRS *)NULL
typedef int  key_type;

/* colors */
#define  VGA_MONOCHROME   0
#define  COLOR            1
#define  BLUE_CYAN     0x31      /* blue on cyan   */
#define  CYAN_BLUE     0x13      /* cyan on blue   */
#define  WHITE_BLUE    0x17      /* white on blue  */
#define  YELLOW_CYAN   0x3E      /* yellow on cyan */
#define  YELLOW_BLUE   0x1E      /* yellow on blue */
#define  BLUE_YELLOW   0xE1      /* blue on yellow */
#define  BWHITE_BLUE   0x1F      /* bright white on blue */
#define  BWHITE_GREEN  0x2F      /* bright white on green*/
#define  WHITE_RED     0x47      /* white on red   */
#define  RED_WHITE     0x74      /* red on white   */
#define  WHITE_BLACK   0x07      /* white on black */
#define  YELLOW_BLACK  0x0E      /* yellow on black */
#define  BLACK_WHITE   0x70      /* black on white */
#define  DGRAY_BLACK   0x08      /* dark gray on black */

/* CSCAPE "pop" messages */
/* message while other process occurs */
#define yellow_message(msg)  pop_Message((msg),-1,-1,-1,-1,YELLOW_BLUE, bd_2)

/* low_message on lower screen while other display on upper screen */
#define low_message(msg)  pop_Message((msg), 20, -1, -1, 30, WHITE_RED, bd_2)

/* shows current forest file name - this can be closed with sed_Close */
#define file_text(msg)    pop_Text((msg), disp_GetHeight()-2, 0, -1, -1, BWHITE_BLUE,
```

```
bd_null)

/* heading */
#define heading_text(msg)  pop_Text((msg), 0, 0, -1, 77, WHITE_BLUE, bd_2)

/* ESC required to exit prompt */
#define prompt(msg)       pop_Prompt ((msg), -1, -1, -1, 40, WHITE_RED, bd_2)

/* unit conversions */
#define FDD_to_CDD              0.5556
#define IN_to_CM                2.54
#define FT_to_CM               30.48
#define SQIN_OZ_SQCM_GM         0.22757
#define AC_to_HA                0.40469
```

```
/********** DATA STRUCTURES ************************/
/* STAND MANAGEMENT data file variables used with linked list */
/* structure for one management year */
typedef struct mgmt_str {
   int mgmt_yr_key;         /*FORTRAN's MYEARS - used as key for linked list */
   int manage;              /*Management Method 1-Target Stem Count 2-Proportion of Stems
*/
   float cutmin[MAX_HOSTS];
   float cutmax[MAX_HOSTS];
   float action[MAX_HOSTS];  /* action may be target or proportion depending on manage */
   struct mgmt_str *next;    /* ptr to the next structure in the linked list */
} MGMT_YRS;


/* DEFOLIATION data file variables used with linked list */
/* structure for one defoliation year */
typedef struct defol_str {
   int defol_yr_key;             /* defoliation yr - used as key */
   int defol_amt[2][MAX_HOSTS];  /* defoliation % ;understory & overstory by host */
   struct defol_str *next;       /* ptr to next structure in list */
} DEFOL_YRS;


/* ISTR structure for stress years */
struct ISTR_str {
   int ISTR;              /* FORTRAN's ISTR - year for stress */
   boolean SELECTED;       /* year user has chosen for stress */
};


/* individual HOST SPECIES data file variables; Struct for each tree species */
typedef struct tree_species_struct {
  char TNAME[3];  /* 2 characters long */
  int host_code;
  int ISHADE[2];
  int ISTOKG;
  int IDHIST[2][2];
  float RESTIN[2];
  float GROWR;
  float COLD;
  float HOT;
  float SLOWD;
  float CR1;
  float CR2;
  float CR3;
  float CR4;
  float DMAX;
  float HMAX;
  float AGEMAX;
  float F1;
  float F2;
  float RECRUT;
  float FDIE[3][3];
  float STEMS[20];
  float SURFAR;
}TREE;
```

```
/********************** DAMAGE MODEL INTERFACE ****************************/
/* Microsoft C 6.0 includes */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>  /* contains declarations for the string handling fncs */
#include <io.h>      /* for access, open functions */
#include <malloc.h>  /* large memory model: returns 4 byte ptrs */
#include <dos.h>     /* _dos_findfirst */
/* CSCAPE 3.2 includes */
#include <cscape.h>
#include <popdecl.h>
#include <framer.h>
#include <helpdecl.h>
#include <teddecl.h>
#include "defines.h"
#include "struct.h"
#include "options.h"   /* #define TEST or COMPLETE */

/* associated object files: x stands for version
mgmtx, defolx, mlinkx, dlinkx, fileiox, hostx, outputx, standx, drawx, pvx,
weatherx, quitx, pcxx, help, helpload */

*****************************************************************************/
/* CSCAPE variables */

/* display parameters for help_View function */
PRIVATE struct hv_struct help_display = {-1,-1,-1, 56, BLACK_WHITE, bd_mouse};

/* CSCAPE window variables */
int win_width,          /* window width */
    win_x,              /* window position (x coord) of upper left hand corner */
    win_y,              /* window position (y coord) of upper left hand corner */
    win_height = 18;    /* window height (18 so that bottom prompt visible) */
sed_type file_heading_sed,  /* used for displaying file */
    heading_sed,     /* upper heading */
    screen_help_sed,  /* displays help line at screen bottom */
    screen_bak,      /* background screen throughout program */
    frame;           /* framer menu */
PRIVATE menu_type  menu_help, /* help line */
    menu_bak;          /* background screen throughout program */
/*****************************************************************************/
/* FOREST MODEL VARIABLES   */


/* GENERAL data file variables; CAPS for consistency with FORTRAN forest model */
char
    STSUBM[2],   /* FORTRAN LOGICAL var; treated as 1 char string */
    GMSUBM[2],
    LRAIN[2],
    TEMPC[2];
int NYEAR,
    IPYEAR,
    IGYEAR,
    NHOSTS,
    ISYEAR,
    IVIEW[MAX_VIEWS],
    ISEED[6],
    NWEAYR,
    IWFORM,
    IWOPT;
float
    TFAC,
    TRETHR;


/* STAND data file variables */
char
    METRIC[2],
    LDEFOL[2];
int ISITE,
    IBOUND,
    ISOPT,
    NSTRYR,
    ICAT;
float
    PLOTAR,
    STNDAR,
    HOVER,
    TLIGHT,
    TKL,
    SHX[6][4],    /* misc. variables */
    SHY[6][4],
```

```
    STRFAC[2],
    PLOTSZ,
    STRESS[3],
    RSMULT,
    STOKX[6][4],
    STOKY[6][4],
    DLENBASE,
    EPSIGR,
    EPSITR,
    EPSIMG[3],
    PRTMIN,
    DFOLDE,
    STOCKS[3][3],
    CRWDMN,
    SHADMN,
    TEMPMN,
    DFLC[2][2],
    SGMORT;


/* STAND MANAGEMENT data file variables used with linked list */
MGMT_YRS *start_mgmt_yr_ptr, mgmt_yr_struct;
int number_of_mgmt_yrs;         /* number of Management Years entered */


/* DEFOLIATION data file variables used with linked list */
DEFOL_YRS *start_defol_yr_ptr, defol_yr_struct;
int number_of_defol_yrs;         /* number of Defoliation Years */


/* WEATHER data file variables */
float
    degree_days[MAX_SIM_YRS];  /* total degree days for one year */


/* array of structures */
struct ISTR_str  ISTR_array[MAX_SIM_YRS];


/* individual HOST SPECIES data file variables */
TREE tree[MAX_HOSTS];  /* array of tree structures */
/*************************************************************************/
/* DAMAGE MODEL OUTPUT variables */


/* output files data structure */
struct output_file_str {
   char output_needed[2];     /* FORTRAN LOGICAL var; treated as 1 char string */
   boolean SAVE_OUTPUT;       /* TRUE if user wants to save output file */
   char description[31];      /* description of output that is needed */
   FILE *outputfile;          /* file created by damage model */
   char model_outputfile_name[13]; /* name of file created by damage model */
   char user_outputfile_name[13];  /* renamed file created by damage model */
};

/* array of structures */
struct output_file_str  outputs[MAX_OUTPUT_FILES];
/*************************************************************************/
/* FILE ACCESS variables;  dos filename - 12 + \n */


/* "species" is read only default file- 20 individual host species data */
char speciesfile_name[13] = "species";

/* forestfile_name is 8 character user defined forest file */
char forestfile_name[9] = " ";
PRIVATE char forestfile_extension[5] = ".inp";
/* long_forestfile_name is forestfile_name + forestfile_extension (".inp") */
char long_forestfile_name[13];
char crash_forestfile_name[13] = "crashed.inp";  /* saved before running model;
                                    insurance for model crash */

/* forest model file names - created for model, then deleted */
PRIVATE char genfile_name[13] =     "gendata.dls";
PRIVATE char standfile_name[13] =   "treedata.dls";
PRIVATE char weatherfile_name[13] = "annddsum.dls";
PRIVATE char defolfile_name[13] =   "defoldat.dls";
PRIVATE char stressfile_name[13] =  "stresyrs.dls";
PRIVATE char notesfile_name[13] =   "usernote.dls";

/* intro file to display */
PRIVATE char introfile_name[13] = "intro";
```

```
FILE *speciesfile, /* 20 host species data */
     *forestfile;   /* general, stand, management, weather, species, description data */
PRIVATE FILE *weatherfile,  /* 6 files created for model, then deleted */
     *genfile,
     *defolfile,   /* defoliation data */
     *stressfile,  /* ISTR values */
     *standfile,
     *notesfile,   /* user text notes */
     *introfile;   /* intro file to display */


int file_options; /* set to 0 after default file read in */
/* file_options individual bits are used to store YES/NO values;
   these flag bits are changed individually with & and |
   bit values of file_options indicate which data is stored in the forest file;
   GEN     0001          0x01      GENERAL
   STAND   0010          0x02      STAND
   WEA     0100          0x04      WEATHER
   STDMAN  1000          0x08      STAND MANAGEMENT
*************************************************************************/
/* FOREST DESCRIPTION variables; defaults loaded 1st;
   21 chars - allow 1 for '\0'*/


char *name = "                      ",
     *job  = "                      ",
     *site = "                      ",
     *date = "                      ";   /* may use CSCAPE date function later */
char user_notes[MAX_NOTE_SIZE];          /* notes used with text editor (ted) */
int line_count = 0;                      /* # of lines in notes file */
/*************************************************************************/
/* USER INTERFACE variables */


int host_to_delete;
boolean CHOICE = FALSE;        /* used in verify_choice fnc */
int tree_index;                /* index of 20 possible hosts (0-19) */
boolean EDITS_MADE = FALSE;    /* editing has been done, prompt to save when exiting*/
boolean EXIT_NOW = FALSE;      /* exit from program when TRUE */
int pcx_shown;                 /* if pcx shown, don't show 2nd intro */
int screen_color;              /* VGA_MONOCHROME or COLOR: read from file */
int command_line_screen_color = COLOR; /* types "m" or "c" on command line */
int units;                     /* = SI or ENGLISH */
int graphics_capable = TRUE;

/* variables used with linked list */
int LOCATED = FALSE;
int status;    /* == SUCCESSFUL_DELETION, KEY_NOT_FOUND, LIST_EMPTY, etc. */
int mouse_works;
/*************************************************************************/
/* USER INTERFACE HOST SPECIES variables */


char *tree_strings[] = {
  "White Oak","Scarlet Oak","Chestnut Oak","Northern Red Oak","Black Oak",
  "Red Maple","Sugar Maple","Striped Maple","Yellow Birch","Paper Birch",
  "Sweet Birch","American Beech","Basswood","Yellow-Poplar","Flowering Dogwood",
  "Quaking Aspen","Black Cherry","White Ash","Hickory", "E. White Pine",
  "Black Gum","User Defined"};

char tree_2chr[POSIBL_SPECIES][3] = {
  "WO","SO","CO","RO","BO",
  "RM","SM","ST","YB","PB",
  "SB","AB","BW","YP","FD",
  "AS","BC","WA","HI","WP",
  "BG","UD" };
/*************************************************************************/
/* FUNCTION PROTOTYPES - fncs called from framer menu need sdata & idata */


/* routines using CSCAPE function calls */
/* sdata & idata needed for framer menu calls */
main(int argc, char *argv[]);
int gen_edit (VOID *sdata, int idata);
int stand_edit(VOID *sdata, int idata);
PRIVATE int user_edit(VOID *sdata, int idata);
PRIVATE void frame_menu (void);
int choose_host_edit(VOID *sdata, int idata);
int delete_species (VOID *sdata, int idata);
int add_species (VOID *sdata, int idata);
void verify_choice ( char msg1[50], char msg2[50] );
int forestfile_edit (void);
```

```
int choose_mgmt_yr_edit (VOID *sdata, int idata);
int add_mgmt_yr (VOID *sdata, int idata);
int delete_mgmt_yr (VOID *sdata, int idata);
int change_mgmt(VOID *sdata, int idata);
int choose_defol_yr_edit (VOID *sdata, int idata);
int add_defol_yr (VOID *sdata, int idata);
int delete_defol_yr (VOID *sdata, int idata);
int select_outputs(VOID *sdata, int idata);
int choose_view_or_print (VOID *sdata, int idata);
int output_to_save (VOID *sdata, int idata);
PRIVATE void set_mono_display (void);
void screen_background(void);
PRIVATE void intro (void);
PRIVATE int choose_inp_file (VOID *sdata, int idata);
PRIVATE int reload_default_file (VOID *sdata, int idata);
int initialize_help(void);
int remove_all_defol (VOID *sdata, int idata);
int weather_edit (VOID *sdata, int idata);
PRIVATE int about_damage(VOID *sdata, int idata);
int choose_output_for_draw (VOID *sdata, int idata);
void screen_help(void);
int init_and_show_pcx(void);
int stand_data(VOID *sdata, int idata);

/* no CSCAPE */
PRIVATE void initialize (void);
void initialize_output(void);
int exit_routine (VOID *sdata, int idata);
PRIVATE int get_file (VOID *sdata, int idata);
PRIVATE int save_file(VOID *sdata, int idata);
void save_forest (char file_name[13]);
void write_gen(FILE *fptr);
void write_stand (FILE * fptr);
void write_weather(FILE *fptr);
void write_mgmt (FILE * fptr);
void write_one_species (FILE *fptr, int index );
void write_STEMS ( FILE *fptr, int index );
void write_defol (FILE * fptr);
void write_ISTR ( FILE *fptr );
void read_forest (char file_name[13]);
PRIVATE void extend_file_name (char long_name[13], char short_name[13], char
extension[5]);
PRIVATE int run_model (VOID *sdata, int idata);
int run_C_model (VOID *sdata, int idata);
PRIVATE void create_model_files(void);
PRIVATE void erase_model_files(void);
void delete_output_after_save(void);
void remove_invalid_chars (char file_name[9]);
PRIVATE void remove_extension (char file_name[9]);
void write_lines_of_notes(FILE *fptr);

/* External FORTRAN subroutine: DAMAGE (STAND) MODEL */
#ifdef FORTRAN_DAMAGE
   extern void fortran damage (void);
#endif
/*****************************************************************************/
/* CSCAPE FRAMER MENU structure */


PRIVATE struct frame_def main_frame[] =
{
  {"   Setup ", NULL, 6},                          /* 1st Main Menu Choice */
    {"Edit job descriptions", user_edit, 0},  /* 1st Pulldown item */
    {"About damage model      ", about_damage, 0},  /* 2nd Pulldown item */
  { FRAME_END },
  {"File ", NULL, 8},                             /* 2nd Main Menu Choice */
    {"Save", save_file, 0},         /* 1st Pulldown item */
    {"Load", get_file, 0},           /* 2nd Pulldown item */
    {"Pick", choose_inp_file, 0},
    {"Reload defaults", reload_default_file, 0},
  { FRAME_END },
  {"Stand ", NULL, 10},                            /* 3rd Main Menu Choice */
    {"General data", gen_edit, 0},                /* 1st Pulldown item */
    {"Stand data     ", stand_data, 0},           /* 2nd Pulldown item */
    {"Stand details  ", stand_edit, 0},
  { FRAME_END },
  {"Trees ", NULL, 12},                       /* 4th Main Menu Choice */
    {"Add tree species", add_species, 0},
    {"Edit tree species", choose_host_edit, 0},
    {"Delete tree species", delete_species, 0},
    {"Add defoliation year", add_defol_yr, 0},
    {"Edit defoliation data", choose_defol_yr_edit, 0},
```

```
    {"Delete defoliation year", delete_defol_yr, 0},
    {"Remove all defoliation ", remove_all_defol, 0},
  { FRAME_END },
  {"Manage ", NULL, 14},                /* 5th Main Menu Choice */
    {"Add management year", add_mgmt_yr, 0},
    {"Edit management data", choose_mgmt_yr_edit, 0},
    {"Delete management year", delete_mgmt_yr, 0},
    {"Change management year", change_mgmt, 0},
  { FRAME_END },
  {"Weather ", NULL, 16},                   /* 6th Main Menu Choice */
    {"Edit weather data", weather_edit, 0},
  { FRAME_END },
  {"Output ", NULL, 18},                    /* 7th Main Menu Choice */
    {"Options", select_outputs, 0},
    {"View   ", choose_view_or_print, 0},
    {"Graph  ", choose_output_for_draw, 0},
    {"Print  ", choose_view_or_print, 1},  /* 1 indicates printing */
    {"Save   ", output_to_save, 0},
  { FRAME_END },
  {"Run ", NULL, 20},                       /* 8th Main Menu Choice */
    {"Run Model", run_model, 0},
  { FRAME_END },
  {"Quit ", exit_routine, 21},              /* 9th Main Menu Choice */
  { FRAME_END },
  { FRAME_END }
};
/*****************************************************************************/


void initialize (void)
/*
DESCRIPTION:
  initialize display interface, load default forest parameters, init weather,
  open display seds
CALLED BY:
  main
CALLS:
  init_and_show_pcx,  read_forest, set_mono_display, screen_background, intro,
  initialize_output, screen_help
NOTES:
  file_options set to 0 after loading default forest file; changed when editing done
*/
{
   int i;
   int scancode;

#ifdef COMPLETE
   pcx_shown = init_and_show_pcx();
#else  /* TEST */
   pcx_shown = SUCCESS;
#endif

#ifdef TEXT_MODE
   if (!disp_Init(def_ModeText, FNULL)) {
      printf("\n Insufficient memory: Unload TSR's (Terminate and Stay Resident
Programs)");
      printf("\n and try again.");
      scancode = getch();
      exit(1);
   }
   graphics_capable = FALSE;
#else  /* GRAPHICS MODE */
   if (!disp_Init(def_ModeGraphics, FNULL)) {
      if (!disp_Init(def_ModeText, FNULL)) {
         printf("\n Insufficient memory: Unload TSR's (Terminate and Stay Resident
Programs)");
         printf("\n and try again.");
         scancode = getch();
         exit(1);
      }
      graphics_capable = FALSE;
   }
#endif

   /* load variables */
   start_mgmt_yr_ptr = NULL_MGMT_PTR;
   start_defol_yr_ptr = NULL_DEFOL_PTR;
   status = NO_DUPLICATE_KEY;

   /* initialize weather */
   for (i = 0; i < MAX_SIM_YRS; i++)
      degree_days[i] = 4570;
```

```
   initialize_output();

   read_forest ("default");
   file_options = 0;

#ifdef TEST
   (void) initialize_help ();
   if (hard_InitMouse())  {
      mouse_works = TRUE;
      sedwin_ClassInit();
   }
   else
      mouse_works = FALSE;
#endif

   /* when we want monochrome, set it */
   if ((screen_color == VGA_MONOCHROME) ||
       (command_line_screen_color == VGA_MONOCHROME) ||
       (disp_GetColors() == 2L))
          set_mono_display();
   if (disp_GetColors() != 2L)  /* if not monochrome, open screen background sed */
      screen_background();

#ifdef COMPLETE
   /* reverse order depending upon whether pcx file shown */
   if (pcx_shown == SUCCESS)   {
      screen_help();
      intro();
   }
   else   {
      intro();
      screen_help();
   }
#else  /* TEST */
   screen_help();
#endif

   /* open heading and file seds */
   heading_sed = heading_text
      ("STAND DAMAGE MODEL: part of the Gypsy Moth Life System Model   Version 1.1");
   file_heading_sed = file_text ("default");
} /* initialize ************************************************/


void set_mono_display (void)
/*
DESCRIPTION:
  translate colors to black and white
CALLED BY:
  initialize
*/
{
   disp_SetMapEntry (0x3E, 0x07, 0x00);   /* yellow on cyan->black on white */
   disp_SetMapEntry (0x0E, 0x07, 0x00);   /* yellow on black->black on white */
   disp_SetMapEntry (0x1E, 0x00, 0x07);   /* yellow on blue->white on black */
   disp_SetMapEntry (0x31, 0x00, 0x07);   /* blue on cyan->white on black */
   disp_SetMapEntry (0x13, 0x00, 0x07);   /* cyan on blue->white on black */
   disp_SetMapEntry (0x47, 0x00, 0x07);   /* white on red->black on white */
   disp_SetMapEntry (0x17, 0x00, 0x07);   /* white on blue->white on black */
   disp_SetMapEntry (0x74, 0x07, 0x00);   /* red on white->white on black */
   disp_SetMapEntry (0x71, 0x07, 0x00);   /* blue on white->black on white */
   disp_SetMapEntry (0x1F, 0x07, 0x00);   /* bright white on blue->black on white */
   disp_SetMapEntry (0x2F, 0x07, 0x00);   /* bright white on green->black on white */
   disp_SetMapEntry (0x1B, 0x00, 0x07);   /* light cyan on blue->white on black */
}  /* set_mono_display ***************************************************/


void intro (void)
/*
DESCRIPTION:
  displays introductions to the program, init mouse
CALLED BY:
  initialize
CALLS:
  initialize_help
*/
{
   static char text[INTRO_BYTES];
   menu_type menu;
   sed_type box_sed;

   /* only display this intro screen if pcx file not shown */
```

```
   if (pcx_shown != SUCCESS)   {
      win_width = 30;
      win_y = 5;
      win_x = 25;
      if ((menu = menu_Open()) == NULL)
         printf("unable to open menu");
      menu_Printf(menu,"@p[1,1]      Stand Damage Model\n");
      menu_Printf(menu,"@p[7,1]     USDA Forest Service\n");
      menu_Printf(menu,"@p[8,1]        Morgantown, WV\n");
      menu_Printf(menu,"@p[9,1]         Version 1.1\n");
      menu_Flush(menu);
      if ((box_sed = sed_Open(menu)) == NULL)
         printf ("unable to open sed");
      sed_SetWidth(box_sed, win_width);
      sed_SetPosition(box_sed, win_y, win_x);
      sed_SetColors (box_sed, BWHITE_GREEN, BWHITE_GREEN, BWHITE_GREEN);
      sed_SetShadow(box_sed, 1);
      sed_SetShadowAttr(box_sed, DGRAY_BLACK);
      sed_Repaint(box_sed);
      sed_Go(box_sed);

      (void) initialize_help ();
      hard_Pause(350);
   }

   /* initialize mouse */
   if (hard_InitMouse())  {
      mouse_works = TRUE;
      sedwin_ClassInit();
   }
   else
      mouse_works = FALSE;

   /* display intro text */
   if ((access(introfile_name, 0)) == 0)  { /* does file exist? */
      introfile = fopen (introfile_name, "r");
      text[fread (text, 1, INTRO_BYTES, introfile)] = '\0';
      if (pcx_shown != SUCCESS)
         sed_Close(box_sed);  /* do it now to avoid gap */
      pop_View (" Introduction ",
                text, 0, 0, disp_GetHeight()-3, 78, YELLOW_BLUE, 1, bd_mouse);
      fclose(introfile);
   }
   else if (pcx_shown != SUCCESS)
      sed_Close(box_sed);
} /* intro ****************************************************/


void screen_help(void)
/*
DESCRIPTION:
  displays help prompt at bottom of screen
CALLED BY:
  initialize, run_model
NOTES:
  screen_help_sed defined global; closed in main function upon exit
*/
{
   int i;
   char msg1[11];
   char msg2[14];
   char msg3[12];

   win_width = disp_GetWidth();
   win_y = disp_GetHeight() - 1;
   win_x = 0;
   strcpy (msg1, "ARROW KEYS");
   strcpy (msg2, "Select Option");
   strcpy (msg3, "Exit Window");

   if ((menu_help = menu_Open()) == NULL)
      printf ("unable to open menu");

   /* separate for monochrome */
   if ((screen_color == VGA_MONOCHROME) ||
       (command_line_screen_color == VGA_MONOCHROME) ||
       (disp_GetColors() == 2L))
          menu_Printf(menu_help,
" F1-Help  %s-Maneuver  ENTER KEY-%s  ESC-%s  Q-Quit",
         msg1, msg2, msg3);

  /* else show colors */
```

```
  else
     menu_Printf(menu_help,
" F1@a[0x70]-Help  @a[0x74]%s@a[0x70]-Maneuver  @a[0x74]ENTER KEY@a[0x70]-%s
@a[0x74]ESC@a[0x70]-%s  @a[0x74]Q@a[0x70]-Quit",
  msg1, msg2, msg3);
   menu_Flush(menu_help);
   if ((screen_help_sed = sed_Open(menu_help)) == NULL)
      printf ("unable to open sed");
   sed_SetPosition(screen_help_sed, win_y, win_x);
   sed_SetWidth(screen_help_sed, win_width);
                          /* grey background          border-nothing */
   sed_SetColors (screen_help_sed, RED_WHITE, RED_WHITE, YELLOW_CYAN);
   sed_SetShadowAttr(screen_help_sed, DGRAY_BLACK);
   sed_SetBorder(screen_help_sed, bd_null);
   sed_Repaint(screen_help_sed);
   sed_Go(screen_help_sed);
} /* screen_help ************************************************/


int about_damage(VOID *sdata, int idata)
/*
DESCRIPTION:
  displays damage information
CALLED BY:
*/
{
   menu_type menu;
   sed_type about_sed;
   int scancode;
   static char text[INTRO_BYTES];

   win_width = 31;
   win_y = 10;
   win_x = 25;
   if ((menu = menu_Open()) == NULL)
      printf ("unable to open menu");
   menu_Printf(menu,"@p[1,1]     Stand Damage Model\n");
   menu_Printf(menu,"@p[6,1]    USDA Forest Service\n");
   menu_Printf(menu,"@p[7,1]       Morgantown, WV\n");
   menu_Printf(menu,"@p[8,1]     Version 1.1     1992\n");
   menu_Printf(menu,"@p[9,1] Press F1 for our address\n");
   menu_Printf(menu,"@p[10,1]Press F2 for more information\n");
   menu_Flush(menu);
   if ((about_sed = sed_Open(menu)) == NULL)
      printf ("unable to open sed");
   sed_SetLabel(about_sed, 2);
   sed_SetMouse(about_sed, sedmou_GreedyClick);
   sed_SetBorder(about_sed, bd_1);
   sed_SetBorderColor(about_sed, YELLOW_BLACK);
   sed_SetWidth(about_sed, win_width);
   sed_SetPosition(about_sed, win_y, win_x);
   sed_SetColors (about_sed, YELLOW_BLACK, YELLOW_BLACK, YELLOW_BLACK);
   sed_SetShadow(about_sed, 1);
   sed_SetShadowAttr(about_sed, 0x08);
   sed_SetExplode(about_sed, exp_BeamMeUp);
   sed_Repaint(about_sed);
   sed_Go(about_sed);

   scancode = kb_Read();   /* wait until keystroke */
   switch (scancode)  {
      case FN1:
          help_Show(sed_GetLabel(about_sed), sed_GetFieldNo(about_sed) + 1);
          break;
      case FN2:
          /* display intro text */
          if ((access(introfile_name, 0)) == 0)  { /* does file exist? */
            introfile = fopen (introfile_name, "r");
            text[fread (text, 1, INTRO_BYTES, introfile)] = '\0';
            pop_View (" Model Information ",
                text, 0, 0, disp_GetHeight()-3, 78, YELLOW_BLUE, 2, bd_mouse);
            fclose(introfile);
          }
          else
            prompt ("Unable to find intro file");
          break;
      default: break;
   }
   sed_Close(about_sed);
   return (0);
} /* about_damage ***********************************************/
```

```
void screen_background(void)
/*
DESCRIPTION:
  displays background color for the screen
CALLED BY:
  initialize, run_model
NOTES:
  menu_bak, screen_bak defined global; closed in main function upon exit;
*/
{
   int i;

   if ((menu_bak = menu_Open()) == NULL)
      printf ("unable to open menu");
   win_y = 0;
   win_x = 0;
   for (i = 0; i < disp_GetHeight(); i++)
      /*display 80 characters: ASCII 176*/
      menu_Printf(menu_bak,"@[%d,%c]\n", disp_GetWidth(), 176);
   menu_Flush(menu_bak);
   if ((screen_bak = sed_Open(menu_bak)) == NULL)
      printf ("unable to open sed");
   sed_SetPosition(screen_bak, win_y, win_x);
   sed_SetColors (screen_bak, 0x70, CYAN_BLUE, 0x07);
   sed_SetShadowAttr(screen_bak, DGRAY_BLACK);
   sed_Repaint(screen_bak);
   sed_Go(screen_bak);
} /* screen_background ************************************************/


int initialize_help(void)
/*
DESCRIPTION:
  initialize help system, load compiled help index
CALLED BY:
  intro, init_and_show_pcx
*/
{
   FILE *help_fp, *index_fp;

   help_fp = fopen("help", "rb");
   if (help_fp == (FILE *) NULL)
   {
     prompt (" Unable to open help file.");
     return (0);
   }

   if (help_Init(help_fp, help_View, HELP_BYTES, (VOID *) &help_display) != HELP_OK)
      prompt(" Unable to initialize help file.");
   else /* now try to load compiled help index */
   {
      if ((index_fp = fopen("help.idx", "r")) != NULL)
      {
         help_LoadTable(index_fp);
         fclose(index_fp);
      }
      else
         prompt(" Unable to load help index");
   }
   return (0);
} /* initialize_help *************************************************/


void frame_menu (void)
/*
DESCRIPTION:
  framer menu
CALLED BY:
  main
NOTES:
  uses CSCAPE framer menu (see struct frame_def main_frame[])
*/
{
   VOID * my_data;

   /* for monochrome, make framer BORDER white on black */
   if ((screen_color == VGA_MONOCHROME) || (disp_GetColors() == 2L))
                                        /*Backgnd,   Selection,  Border */
      frame = frame_Open(main_frame, bd_1, BLUE_CYAN, BWHITE_BLUE, WHITE_BLACK);
   else
      frame = frame_Open(main_frame, bd_1, BLUE_CYAN, BWHITE_BLUE, YELLOW_CYAN);
   frame_SetPosition (frame, 3, 0);
```

```
    sed_SetLabel (frame, 5);
    sed_SetShadowAttr(frame, DGRAY_BLACK);
    frame_Repaint(frame);
    frame_Go(frame, ' ', (VOID *) my_data);
    frame_Close(frame);
} /* frame_menu ************************************************************/


int user_edit(VOID *sdata, int idata)
/*
DESCRIPTION:
  edits job headings for forest file
CALLED BY:
  framer menu choice EDIT JOB DESCRIPTIONS
CALLS:
  remove_invalid_chars, remove_extension, extend_file_name
NOTES:
  contains inner and outer windows
*/
{
   menu_type out_menu, inner_menu;
   sed_type out_sed;           /* outer sed for file name, date, etc. */
   sed_type inner_ted_sed;     /* text editor (ted) for entering notes */
   bob_type bob;
   int in_win_width;
   int in_win_height;

   EDITS_MADE = TRUE;
   /* create inner sed */
   if ((inner_menu = menu_Open()) == NULL)
   {
      printf ("unable to open menu");
      return(0);
   }
   win_width = 76;
   in_win_width = 74;
   in_win_height = 7;
   win_y = 7;
   win_x = 3;
   menu_Printf(inner_menu, "@f[]",
     user_notes, &ted_funcs);
   menu_Flush(inner_menu);
   if ((inner_ted_sed = sed_Open(inner_menu)) == NULL)
   {
      printf ("unable to open sed");
      return(0);
   }
   ted_SetMaxSize(inner_ted_sed, 500L);
   ted_SetWrapWidth (inner_ted_sed, LINE_SIZE);
   sed_SetLabel(inner_ted_sed, 49);
   sed_SetWidth(inner_ted_sed, in_win_width);
   sed_SetHeight(inner_ted_sed, in_win_height);
   sed_SetPosition(inner_ted_sed, win_y, win_x);
   sed_SetColors (inner_ted_sed, YELLOW_CYAN, YELLOW_CYAN, BLUE_YELLOW);
   sed_SetSpecial(inner_ted_sed, spc_EmbedTed);
   bob = sed_CreateBob(inner_ted_sed, BOB_DEPENDENT);
   sed_SetMouse(inner_ted_sed, sedmou_GreedyTrack);
   sed_SetBorder(inner_ted_sed, bd_mouse);

   /* create outer sed */
   if ((out_menu = menu_Open()) == NULL)
   {
      printf ("unable to open menu");
      return(0);
   }
   menu_Printf(out_menu, "Disk File Name: @fd[########]\n",
     forestfile_name, &string_funcs,"DOS File Name (1-8 alphanumerics; NO blanks or
periods)");
   menu_Printf(out_menu, "     User Name: @fd[####################]\n",
     name, &string_funcs,"Enter User Name");
   menu_Printf(out_menu, "     Job Title: @fd[####################]\n",
     job, &string_funcs,"Enter Job Title for this Project");
   menu_Printf(out_menu, " Site Location: @fd[####################]\n",
     site, &string_funcs,"Enter Site Location");
   menu_Printf(out_menu, "          Date: @fd[####################]\n",
     date, &string_funcs,"Enter Project/Job Date");
   menu_Printf(out_menu, "%s\n", "  @a[0x1E]Run description text (CNTRL-PAGEUP to exit run
description subscreen)");
   menu_Printf(out_menu, "@fb[]", NULL, &bob_funcs, bob);
   menu_Flush(out_menu);
   if ((out_sed = sed_Open(out_menu)) == NULL)
   {
```

```
      printf ("unable to open sed");
      return(0);
   }
   sed_SetLabel(out_sed, 22);
   sed_SetWidth(out_sed, win_width);
   sed_SetPosition(out_sed, win_y, win_x);
   sed_SetColors (out_sed, CYAN_BLUE, WHITE_BLUE, YELLOW_CYAN);
   sed_SetMouse(out_sed, sedmou_GreedyTrack);
   sed_SetBorder(out_sed, bd_std);
   sed_SetBorderTitle(out_sed, "                                USER INFORMATION");
   sed_Repaint(out_sed);
   sed_Go(out_sed);
   sed_Close(out_sed);                  /* also closes inner_ted_sed and bob*/
   if (strcmp(" ", forestfile_name)!=0)    /* user has entered name */
   {
      remove_invalid_chars (forestfile_name);
      remove_extension (forestfile_name);
      extend_file_name(long_forestfile_name, forestfile_name, forestfile_extension);
      sed_Close (file_heading_sed);    /* close old file heading sed */
      file_heading_sed = file_text (long_forestfile_name);
   }
   return (0);
} /* user_edit ************************************************************/


void remove_invalid_chars (char file_name[9])
/*
DESCRIPTION:
  removes invalid chars
ARGUMENTS:
  DOS file name
RETURNS:
  file_name without invalid_chars
CALLED BY:
  file_edit, user_edit, create_outputfile_name
SAMPLE CALL:
  remove_invalid_chars ( " file1" );
*/
{
   char valid_chars[NUM_VALID_CHARS] =
      "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789_-.:\0";
   int i,k;
   int j = 0;
   int len;
   char temp_chrs[13];

   len = strlen(file_name);
   /* get rid of invalid chars */
   for ( i=0; i < (len+1); i++ )
      for ( k=0; k < NUM_VALID_CHARS; k++ )
         if (file_name[i] == valid_chars[k])
         {
            temp_chrs[j] = file_name[i];
            file_name[j] = temp_chrs[j];
            j = j+1;
         }
} /* remove_invalid_chars ************************************************/


void remove_extension (char file_name[9])
/*
DESCRIPTION:
  removes everything past the decimal
ARGUMENTS:
  DOS file name
RETURNS:
  file_name without extension
CALLED BY:
  file_edit, user_edit
SAMPLE CALL:
  remove_extension ( " fil.e1" );
*/
{
   int i;
   int len;
   char temp_name[9];

   /* get rid of everything after period */
   len = strlen(file_name);
   for ( i=0; i < (len+1); i++ )
      if (file_name[i] == '.')
      {
```

```
         beep();
         prompt("  Extension will be removed");
         strncpy (temp_name, file_name, i);
         temp_name[i] = '\0';
         strcpy(file_name, temp_name);
      }
} /* remove_extension ***************************************************/


void extend_file_name (char long_name[13], char short_name[9], char extension[5])
/*
DESCRIPTION:
  extends file name (8 char file name + 3 char extension)
CALLED BY:
  user_edit, forestfile_edit
ARGUMENTS:
  extension is 3 char DOS extension; short_name is DOS file name without extension
  long_name is DOS file name + extension
SAMPLE CALL
  extend_file_name(long_forestfile_name, forestfile_name, forestfile_extension);
*/
{
  strcpy (long_name, short_name);
  strcat (long_name, extension);
} /* extend_file_name ***************************************************/


int save_file(VOID *sdata, int idata)
/*
DESCRIPTION:
  saves file to disk
CALLED BY:
  framer menu SAVE INPUT FILE
CALLS:
  forestfile_edit, verify_choice, save_forest
*/
{
   boolean FILE_SAVED = FALSE;
   int count = 0;  /* count of number of attempts */
   char file_msg_string[30];

   while ((FILE_SAVED == FALSE) && (count < 2))
   {
      if (strcmp(" ", forestfile_name)==0)    /* user has not entered name */
      {
         count++;
         low_message("No name entered - Enter disk file name (ESC to Abort)");
         (void) forestfile_edit ();
         low_message(NULL);
      }
      else if ((access(long_forestfile_name, 0)) == 0) /* file exist? */
      {
         count++;
         CHOICE = FALSE;
         verify_choice ( "OVERWRITE existing File?",
                         "WARNING: File exists on disk");
         if (CHOICE == TRUE)
         {
            /* overwrite file */
            sprintf (file_msg_string, "Saving %s", long_forestfile_name);
            yellow_message (file_msg_string);
            save_forest (long_forestfile_name);
            FILE_SAVED = TRUE;
            EDITS_MADE = FALSE;
            hard_Pause (DELAY);
            yellow_message (NULL);
         }
         else
         {
            low_message("Enter NEW Disk File Name");
            (void) forestfile_edit ();
            low_message(NULL);
         }
      }
      else
      {
         /* not overwriting file */
         sprintf (file_msg_string, "Saving %s", long_forestfile_name);
         yellow_message (file_msg_string);
         save_forest (long_forestfile_name);
         FILE_SAVED = TRUE;
         EDITS_MADE = FALSE;
```

```
         hard_Pause (DELAY);
         yellow_message (NULL);
      }
   }  /* while */
   if ((FALSE == FILE_SAVED) && (2 == count))
   {
      beep();
      prompt ("File Not Saved");
   }
   return (0);
} /* save_file ********************************************************/


int forestfile_edit(void)
/*
DESCRIPTION:
  edits forest file name
CALLED BY:
  exit_routine, get_file
CALLS:
  remove_invalid_chars, remove_extension, extend_file_name
*/
{
   menu_type menu;
   sed_type file_sed;

   if ((menu = menu_Open()) == NULL)
   {
      printf ("unable to open menu");
      return(0);
   }
   win_width = 75;
   win_y = 11;
   win_x = 3;
   menu_Printf(menu, "Disk File Name: @fd[########]\n",
     forestfile_name, &string_funcs,"DOS File Name (1-8 alphanumerics; NO blanks or
periods)");
   menu_Flush(menu);
   if ((file_sed = sed_Open(menu)) == NULL)
   {
      printf ("unable to open sed");
      return(0);
   }
   sed_SetLabel(file_sed, 23);
   sed_SetPosition(file_sed, win_y, win_x);
   sed_SetColors (file_sed, CYAN_BLUE, WHITE_BLUE, YELLOW_CYAN);
   sed_SetShadow(file_sed, 1);
   sed_SetWidth(file_sed, win_width);
   sed_SetMouse(file_sed, sedmou_GreedyClick);
   sed_SetBorder(file_sed, bd_std);
   sed_SetBorderTitle(file_sed, "    DISK FILE NAME     ");
   sed_Repaint(file_sed);
   sed_Go(file_sed);
   sed_Close(file_sed);
   if (strcmp(" ", forestfile_name)!=0)    /* user has entered name */
   {
      remove_invalid_chars (forestfile_name);
      remove_extension (forestfile_name);
      extend_file_name(long_forestfile_name, forestfile_name, forestfile_extension);
      sed_Close (file_heading_sed);    /* close old file heading sed */
      file_heading_sed = file_text (long_forestfile_name);
   }
   return (0);
} /* forestfile_edit ********************************************************/


/***********************************************/
void verify_choice (char msg1[50], char msg2[50])
/***********************************************
DESCRIPTION:verify choice user is making
ARGUMENTS:msg1 and msg2 are message prompts
SAMPLE CALL:verify_choice ("SAVE File before EXIT?","WARNING: Edited data may be lost");
CALLED BY:exit_routine, get_file, save_file, delete_mgmt_yr, delete_defol_yr,
remove_all_defol
NOTES:CHOICE is global boolean; CHOICE set to TRUE or FALSE before call
*/
{
   menu_type menu;
   sed_type sed;
   int ret;

   if ((menu = menu_Open()) == NULL)
```

```
      printf ("unable to open menu");
   win_width = max(strlen(msg1), strlen(msg2)) + 2;
   win_y = disp_GetHeight() - 10;
   win_x = 18;
   menu_Printf(menu, "\n %s \n", msg2 );
   menu_Printf(menu, " %s\n", msg1 );
   menu_Printf(menu,"@[76,%c]\n", 196);
   menu_Printf (menu,"@[8, ]@f[  Yes  ] ",NULL,&gmenu_funcs);
   menu_Printf (menu,"   @f[  No   ] ",NULL,&gmenu_funcs);
   menu_Flush(menu);
   if ((sed = sed_Open(menu)) == NULL)
      printf ("unable to open sed");
   sed_SetLabel(sed, 24);
   sed_SetWidth(sed, win_width);
   sed_SetPosition(sed, win_y, win_x);
   sed_SetColors (sed, 0x47, WHITE_RED, BLACK_WHITE);
   sed_SetShadow(sed, 1);
   sed_SetShadowAttr(sed, DGRAY_BLACK);
   sed_SetMouse(sed, sedmou_GreedyTrack);
   sed_SetBorder(sed, bd_mouse);
   sed_SetBorderTitle(sed, " Message ");
   if (CHOICE == FALSE)
      sed_GotoField(sed, 1);
   sed_Repaint(sed);
   ret = sed_Go(sed);
   sed_Close(sed);
   if (1 == ret)
      CHOICE = TRUE;
   else if (2 == ret)
      CHOICE = FALSE;
} /* verify_choice ********************************************************/


int get_file (VOID *sdata, int idata)
/*
DESCRIPTION:
  gets file from disk
CALLED BY:
  framer menu GET INPUT FILE
CALLS:
  forestfile_edit, verify_choice, read_forest, delete_output_after_save
NOTES:
  2 exit points
  if file does not exist and new name entered, new name remains
  deletes mgmt_yr and defol_yr linked lists
*/
{
  boolean EXIT_ROUTINE = FALSE;
  MGMT_YRS  *temp_mgmt_ptr;
  DEFOL_YRS *temp_defol_ptr;
  char file_msg_string[30];

  delete_output_after_save();
  if (EDITS_MADE == TRUE)
  {
     CHOICE = TRUE;
     verify_choice ( "SAVE File before getting new file?",
                     "WARNING: Edited data may be lost");
     if (CHOICE == TRUE)
        return (0);  /* exit */
  }
  while (EXIT_ROUTINE == FALSE)
  {
     low_message("Enter NEW Disk File to Retrieve");
     (void) forestfile_edit ();
     low_message(NULL);
     if ((access(long_forestfile_name, 0)) == 0) /* does file exist? */
     {
        sprintf (file_msg_string, "Loading %s", long_forestfile_name);
        yellow_message (file_msg_string);

        /* read in defaults to reset variables */
        read_forest ("default");

        /* delete all old structures in linked list; free memory */
        while (start_mgmt_yr_ptr != NULL_MGMT_PTR)
        {
           /* use temp_mgmt_ptr to dispose of unwanted str */
           temp_mgmt_ptr = start_mgmt_yr_ptr;
           start_mgmt_yr_ptr = start_mgmt_yr_ptr->next;
           free (temp_mgmt_ptr);
        }
```

```
      start_mgmt_yr_ptr = NULL_MGMT_PTR;
      number_of_mgmt_yrs = 0;

      while (start_defol_yr_ptr != NULL_DEFOL_PTR)
      {
         /* use temp_defol_ptr to dispose of unwanted str */
         temp_defol_ptr = start_defol_yr_ptr;
         start_defol_yr_ptr = start_defol_yr_ptr->next;
         free (temp_defol_ptr);
      }
      start_defol_yr_ptr = NULL_DEFOL_PTR;
      number_of_defol_yrs = 0;

      status = NO_DUPLICATE_KEY;
      EDITS_MADE = FALSE;
      EXIT_ROUTINE = TRUE;
      line_count = 0;
      read_forest (long_forestfile_name);
      hard_Pause (DELAY);
      yellow_message (NULL);
   }
   else
   {
      CHOICE = TRUE;
      verify_choice ( "Do you wish to enter new file name?",
                  "FILE not found on disk!");
      if (CHOICE == FALSE)
         EXIT_ROUTINE = TRUE;
      /* else allow user to enter new name */
   }
  } /* while */
  return (0);
} /* get_file ******************************************************/


int choose_inp_file (VOID *sdata, int idata)
/*
DESCRIPTION:
  display input files (all files with .inp extension) and get one
CALLED BY:
  framer_menu CHOOSE INPUT FILE
CALLS:
  verify_choice, read_forest, delete_output_after_save
NOTES:
  routine has 2 exit points
*/
{
  menu_type menu;
  sed_type sed;
  int i;
  int line_counter=1;
  int ret, n;
  struct find_t  inp_file;    /* dos.h */
  char menu_inp_string[20];   /* "@f[inp_file.name]\n" */
  MGMT_YRS  *temp_mgmt_ptr;
  DEFOL_YRS *temp_defol_ptr;
  char file_msg_string[30];

  delete_output_after_save();
  /* find first .inp file in current directory */
  if (_dos_findfirst ("*.inp", _A_NORMAL, &inp_file)  == 0) {
     if (EDITS_MADE == TRUE) {
        CHOICE = TRUE;
        verify_choice ( "SAVE File before getting new file?",
                        "WARNING: Edited data may be lost");
        if (CHOICE == TRUE)
           return (0);  /* exit */
     }
     if ((menu = menu_Open()) == NULL)  {
        printf ("unable to open menu");
        return(0);
     }
     win_width = 25;
     win_y = 5;
     win_x = 31;

     sprintf (menu_inp_string, " @f[%s]\n", inp_file.name);
     menu_Printf(menu, menu_inp_string, NULL, &menu_funcs);
     while (_dos_findnext(&inp_file) == 0)  {
        line_counter++;
        sprintf (menu_inp_string, " @f[%s]\n", inp_file.name);
```

```
         menu_Printf(menu, menu_inp_string, NULL, &menu_funcs);
      }
      menu_Flush(menu);
      if ((sed = sed_Open(menu)) == NULL)  {
         printf ("unable to open sed");
         return(0);
      }
      sed_SetLabel(sed, 25);
      sed_SetWidth(sed, win_width);
      sed_SetHeight(sed, min(line_counter, disp_GetHeight()-7));
      sed_SetPosition(sed, win_y, win_x);
      sed_SetColors (sed, CYAN_BLUE, WHITE_BLUE, YELLOW_CYAN);
      sed_SetShadow(sed, 1);
      sed_SetShadowAttr(sed, DGRAY_BLACK);
      sed_SetMouse(sed, sedmou_GreedyTrack);
      sed_SetBorder(sed, bd_mouse);
      sed_SetBorderTitle(sed, " Choose file with ENTER");
      sed_Repaint(sed);
      ret=sed_Go(sed);
      sed_Close(sed);
      if (ret > 0) {
         _dos_findfirst ("*.inp", _A_NORMAL, &inp_file);
         if (ret > 1)
            for (n=1; n < ret; ++n)
               _dos_findnext(&inp_file);
         strcpy (long_forestfile_name, inp_file.name);
         strncpy (forestfile_name, long_forestfile_name, (strlen(long_forestfile_name)-4));
         forestfile_name[strlen(long_forestfile_name)-4] = '\0';
         sed_Close (file_heading_sed);    /* close old file heading sed */
         file_heading_sed = file_text (long_forestfile_name);

         sprintf (file_msg_string, "Loading %s", long_forestfile_name);
         yellow_message (file_msg_string);
         /* read in defaults to reset variables */
         read_forest ("default");

         /* delete all old structures in linked list; free memory */
         while (start_mgmt_yr_ptr != NULL_MGMT_PTR) {
            /* use temp_mgmt_ptr to dispose of unwanted str */
            temp_mgmt_ptr = start_mgmt_yr_ptr;
            start_mgmt_yr_ptr = start_mgmt_yr_ptr->next;
            free (temp_mgmt_ptr);
         }
         start_mgmt_yr_ptr = NULL_MGMT_PTR;
         number_of_mgmt_yrs = 0;

         while (start_defol_yr_ptr != NULL_DEFOL_PTR) {
            /* use temp_defol_ptr to dispose of unwanted str */
            temp_defol_ptr = start_defol_yr_ptr;
            start_defol_yr_ptr = start_defol_yr_ptr->next;
            free (temp_defol_ptr);
         }
         start_defol_yr_ptr = NULL_DEFOL_PTR;
         number_of_defol_yrs = 0;

         status = NO_DUPLICATE_KEY;
         EDITS_MADE = FALSE;
         line_count = 0;
         read_forest (long_forestfile_name);
         hard_Pause (DELAY);
         yellow_message (NULL);
      }
   }
   else
      prompt ("NO *.inp files found");
   return (0);
} /* choose_inp_file ************************************************/


int reload_default_file (VOID *sdata, int idata)
/*
DESCRIPTION:
  reloads default file from disk
CALLED BY:
  framer menu RELOAD DEFAULT FILE
CALLS:
  verify_choice, read_forest, delete_output_after_save
NOTES:
  2 exit points
  deletes mgmt_yr and defol_yr linked lists
*/
{
```

```
  MGMT_YRS  *temp_mgmt_ptr;
  DEFOL_YRS *temp_defol_ptr;
  int i;

  delete_output_after_save();
  if (EDITS_MADE == TRUE)
  {
     CHOICE = TRUE;
     verify_choice ( "SAVE File before loading default data?",
                     "WARNING: Edited data may be lost");
     if (CHOICE == TRUE)
        return (0);  /* exit */
  }

  yellow_message("Loading Default File");
  /* delete all old structures in linked list; free memory */
  while (start_mgmt_yr_ptr != NULL_MGMT_PTR)
  {
     /* use temp_mgmt_ptr to dispose of unwanted str */
     temp_mgmt_ptr = start_mgmt_yr_ptr;
     start_mgmt_yr_ptr = start_mgmt_yr_ptr->next;
     free (temp_mgmt_ptr);
  }
  start_mgmt_yr_ptr = NULL_MGMT_PTR;
  number_of_mgmt_yrs = 0;

  while (start_defol_yr_ptr != NULL_DEFOL_PTR)
  {
      /* use temp_defol_ptr to dispose of unwanted str */
      temp_defol_ptr = start_defol_yr_ptr;
      start_defol_yr_ptr = start_defol_yr_ptr->next;
      free (temp_defol_ptr);
  }
  start_defol_yr_ptr = NULL_DEFOL_PTR;
  number_of_defol_yrs = 0;

  /* initialize weather */
  for (i = 0; i < MAX_SIM_YRS; i++)
     degree_days[i] = 4570;

  /* read in default file */
  read_forest ("default");

  status = NO_DUPLICATE_KEY;
  EDITS_MADE = FALSE;
  line_count = 0;
  file_options = 0;
  strcpy (forestfile_name, " ");
  strcpy (long_forestfile_name, " ");
  sed_Close (file_heading_sed);
  hard_Pause (DELAY);
  yellow_message (NULL);
  file_heading_sed = file_text ("default");
  return (0);
} /* reload_default_file ***************************************************/


int run_model (VOID *sdata, int idata)
/*
DESCRIPTION:
  delete existing output, create model & crash forest files, run model,
  set output variables, delete model & crash forest files
CALLED BY:
  framer menu RUN MODEL
CALLS:
  create_model_files, save_forest, damage (FORTRAN MODEL), erase_model_files,
  screen_background, delete_output_after_save, screen_help
NOTES:
  temporary forest file created in case of fatal error in forest model
*/
{
   int i;
   int result;

   (void) delete_output_after_save();
   yellow_message ("Model is loading ... Please Standby");
   create_model_files ();
   save_forest (crash_forestfile_name);
   yellow_message (NULL);

   /* close sed windows to save memory while running model */
   if (disp_GetColors() != 2L)  /* if not monochrome, close background sed */
```

```
      sed_Close(screen_bak);
   sed_Close (file_heading_sed);     /* close file heading sed */
   sed_Close (heading_sed);          /* close heading sed */
   sed_Close (screen_help_sed);      /* close help prompt sed */
   sed_Pop (frame);                  /* temp removal of frame menu */
   disp_HideMouse();

#ifdef FORTRAN_DAMAGE
   damage();           /* DAMAGE.OBJ: FORTRAN stand model */
#endif

   /* set output file's parameters */
   outputs[STABLE].SAVE_OUTPUT = FALSE;
   for (i = 1; i < MAX_OUTPUT_FILES; i++)  /* i = 1 skips STABLE - 0 */
   {
      /* does file exist AND user set output to TRUE? */
      if (((access(outputs[i].model_outputfile_name, 0)) == 0) &&
           ((strcmp("T", outputs[i].output_needed)==0) || (strcmp("t",
outputs[i].output_needed)==0)))
              outputs[i].SAVE_OUTPUT = TRUE;
      else
         outputs[i].SAVE_OUTPUT = FALSE;
   }
   /* reset user_outputfile_name so new names can be used */
   for (i = 0; i < MAX_OUTPUT_FILES; i++)
      strcpy (outputs[i].user_outputfile_name, " ");

   /* reopen sed windows after running model */
   if (disp_GetColors() != 2L)
      screen_background();
   heading_sed = heading_text
      ("STAND DAMAGE MODEL: part of the Gypsy Moth Life System Model   Version 1.1");

   if (strcmp(" ", forestfile_name)!=0)    /* user has entered name */
      file_heading_sed = file_text (long_forestfile_name);
   else
      file_heading_sed = file_text ("default");
   screen_help();

   sed_Repaint (frame);
   disp_ShowMouse();

   result = remove(crash_forestfile_name);
   if (result == -1)
     printf ("%s file could not be opened.\n", crash_forestfile_name );

   erase_model_files ();

   return (0);
} /* run_model ******************************************************/


void create_model_files(void)
/*
DESCRIPTION:
  creates model files
CALLED BY:
  run_model
CALLS:
  write_gen, write_stand, write_weather, write_mgmt, write_one_species,
  write_STEMS, write_defol, write_ISTR, write_lines_of_notes
*/
{
  int  i;
  int note_length;     /* length of user_notes */

  genfile = fopen (genfile_name, "w");
  if (genfile == (FILE *) NULL )
     printf ("%s file could not be opened.\n", genfile_name);

  note_length = strlen (user_notes);
  if (note_length > 0)
  {
     notesfile = fopen (notesfile_name, "w");
     if (notesfile == (FILE *) NULL )
        printf ("%s file could not be opened.\n", notesfile_name);
  }
  standfile = fopen (standfile_name, "w");
  if (standfile == (FILE *) NULL )
     printf ("%s file could not be opened.\n", standfile_name);
  /* do defoliation years exist? */
  if (start_defol_yr_ptr != NULL_DEFOL_PTR)
```

```
  {
     defolfile = fopen (defolfile_name, "w");
     if (defolfile == (FILE *) NULL )
        printf ("%s file could not be opened.\n", defolfile_name);
  }
  weatherfile = fopen (weatherfile_name, "w");
  if (weatherfile == (FILE *) NULL )
     printf ("%s file could not be opened.\n", weatherfile_name);
  /* do stress years exist? */
  if (NSTRYR > 0)
  {
     stressfile = fopen (stressfile_name, "w");
     if (stressfile == (FILE *) NULL )
        printf ("%s file could not be opened.\n", stressfile_name);
  }

  /********* genfile *********/
  fprintf(genfile, " %2d", NHOSTS);

  /******** notesfile ********/
  line_count = 0;
  if (note_length > 0)
     write_lines_of_notes (notesfile);

  /********* genfile *********/
  fprintf(genfile, " %2d\n", line_count);  /* this must follow write_lines_of_notes*/
  if (strcmp(" ", forestfile_name)!=0)    /* user has entered name */
     fprintf(genfile, "%s\n", long_forestfile_name);
  else  /* user using default file */
     fprintf(genfile, "default\n");
  fprintf(genfile, "%s\n", name);
  fprintf(genfile, "%s\n", job );
  fprintf(genfile, "%s\n", site);
  fprintf(genfile, "%s\n", date);
  write_gen (genfile);

  /******** standfile ********/
  write_stand (standfile);
  for ( i=0; i < NHOSTS; i++ )
  {
     write_one_species ( standfile, i );
     write_STEMS (standfile, i);
  }
  write_mgmt (standfile);

  /******** weatherfile ******/
  write_weather (weatherfile);

  /******** defolfile ********/
  if (start_defol_yr_ptr != NULL_DEFOL_PTR)
     write_defol (defolfile);

  /******** stressfile ******/
  if (NSTRYR > 0)
     write_ISTR (stressfile);

  fclose(genfile);
  if (note_length > 0)
     fclose(notesfile);
  fclose(standfile);
  if (start_defol_yr_ptr != NULL_DEFOL_PTR)
     fclose(defolfile);
  fclose(weatherfile);
  if (NSTRYR > 0)
     fclose(stressfile);
} /* create_model_files ****************************************************/


void erase_model_files(void)
/*
DESCRIPTION:
  erases model files - gen, defol, weather, stress
CALLED BY:
  run_model;
*/
{
  int result;
  int note_length;     /* length of user_notes */

  result = remove(genfile_name);
  if (result == -1)
     printf ("%s file could not be opened.\n", genfile_name );
```

```
   note_length = strlen (user_notes);
   if (note_length > 0)
   {
      result = remove(notesfile_name);
      if (result == -1)
         printf ("%s file could not be opened.\n", notesfile_name );
   }
   result = remove(standfile_name);
   if (result == -1)
      printf ("%s file could not be opened.\n", standfile_name );
   if (start_defol_yr_ptr != NULL_DEFOL_PTR)
   {
      result = remove(defolfile_name);
      if (result == -1)
         printf ("%s file could not be opened.\n", defolfile_name );
   }
   result = remove(weatherfile_name);
   if (result == -1)
      printf ("%s file could not be opened.\n", weatherfile_name );
   if (NSTRYR > 0)
   {
      result = remove(stressfile_name);
      if (result == -1)
         printf ("%s file could not be opened.\n", stressfile_name );
   }
} /* erase_model_files ****************************************************/


main(int argc, char *argv[])
/*
DESCRIPTION:
  main program control
CALLS:
  initialize, frame_menu
ARGUMENTS:
  argv[1] can be command for monochrome, i.e., "damage m"
*/
{
   if (argc == 2) /* command line argument, check for mono "m" or color "c" */
      if ((strcmp(argv[1], "m")==0) || (strcmp(argv[1], "M")==0))
         command_line_screen_color = VGA_MONOCHROME;

   initialize ();
   while (!EXIT_NOW)
      frame_menu ();

   if (disp_GetColors() != 2L)  /* if not monochrome, close background sed */
      sed_Close(screen_bak);

   sed_Close (file_heading_sed);     /* close file heading sed */
   sed_Close (screen_help_sed);
   sed_Close (heading_sed);          /* close heading sed */

   disp_Close();
   exit(0);
   return(0);
} /* main ***************************************************************/
```

```
/************************* PCX LOGO DISPLAY ********************************/
/* Open pmap (pixel map graphics) window, load image from .PCX file */
/* Microsoft C 6.0 includes */
#include <stdio.h>
#include <string.h>
#include <io.h>      /* for access */
#include <graph.h>
/* CSCAPE 3.2 includes */
#include <cscape.h>
#include <pmwinobj.h>
#include <popdecl.h>
/* define file */
#include "defines.h"

/*** Function prototypes ***/
int init_and_show_pcx (void);
boolean display_pcx(char *fname);
int initialize_help(void);
void verify_choice ( char msg1[50], char msg2[50] );


/*** Global data ***/
PRIVATE win_type pcxwin;
PRIVATE sed_type back_sed;
PRIVATE menu_type back_menu;
extern boolean CHOICE;


int init_and_show_pcx (void)
/*
DESCRIPTION:
  displays pcx image
CALLED BY:
  initialize
CALLS:
  display_pcx, initialize_help
NOTES:
  has 4 exits; can exit program
*/
{
   ocbox charbox;
   int i;
   char pcx_filename[13] = "oak.pcx";
   int FAILURE = 1;

   /* can display EGA? */
   if (!pc_IsEGA())
      return(FAILURE);

   /* pcx file on disk? */
   if ((access(pcx_filename, 0)) != 0)
      return(FAILURE);

   /* does monitor have graphics capability? */
   if (!disp_Init(def_ModeGraphics, FNULL))
      return(FAILURE);

   /* Turn on pmap window .pcx file handling */
   pmap_IoInit();

   /* pcx image window */
   if (pc_IsVGA())
   {  /* VGA is 80x30 char: 640x480 */
      charbox.toprow   = 4;
      charbox.leftcol  = 20;
      charbox.botrow   = 22;
      charbox.rightcol = 58;
   }
   else  /* must be EGA: 80x25 char: 640x350; need vertical space */
   {
      charbox.toprow   = 1;
      charbox.leftcol  = 20;
      charbox.botrow   = 22;
      charbox.rightcol = 59;
   }

   pcxwin = win_Open(pmwin_Class, &charbox);

   /* draw background screen *****/
   if ((back_menu = menu_Open()) == NULL)
      printf ("unable to open back_menu");
   for (i = 0; i < disp_GetHeight(); i++)
```

```
      /*display 80 characters: ASCII 176*/
      menu_Printf(back_menu,"@[%d,%c]\n", disp_GetWidth(), 176);
   menu_Flush(back_menu);
   if ((back_sed = sed_Open(back_menu)) == NULL)
      printf ("unable to open back_sed");
   sed_SetPosition(back_sed, 0, 0);
   sed_SetColors (back_sed, 0x70, CYAN_BLUE, 0x07);
   sed_SetShadowAttr(back_sed, DGRAY_BLACK);

   /* display pcx file */
   if (!display_pcx(pcx_filename)) {
      /* FAILURE */
      win_Close(pcxwin);
      CHOICE = FALSE;
      verify_choice ( "Continue STAND-DAMAGE program?",
                                    "WARNING: Memory Limit: Cannot load graphic!");
      if (CHOICE == FALSE) {
                disp_Close(); /* close graphic display interface */
                exit(0);
                return(0);
      }
      disp_Close(); /* close graphic display interface */
      return(FAILURE);
   }
   else
   {
      (void) initialize_help(); /* helps Pause */
      hard_Pause(250);
      sed_Close(back_sed);
      win_Close(pcxwin);
      disp_Close(); /* close graphic display interface */
      return(SUCCESS);
   }
} /* init_and_show_pcx *****************************************************/


boolean display_pcx(char *fname)
/*
DESCRIPTION:
  open .pcx file and show the image
ARGUMENTS:
  fname is pcx file
CALLED BY:
  init_and_show_pcx
CALLS:
*/
{
  FILE *fp;
  pmap_type pmap = NULL;
  ocolmap_type crange;

  /* First, try to open file in binary mode */
  if ((fp = fopen(fname, "rb")) != NULL)
  {
     /* Create color range for pmap; color range defines color palette
                used to create image */
     crange = ocolmap_Open(0, 16);

     /* Load pmap from file and close file */
     pmap = pmap_Load(fp, crange);
     fclose(fp);

     ocolmap_Close(crange);   /* close color range */
     pmwin_SetPmap(pcxwin, pmap);   /* attach pmap to window */
  }
  if (pmap != NULL)
  {
     win_SetBorder(pcxwin, bd_2);
     /* set window background color = pcx color so that they blend together */
     bord_SetAttr (pcxwin, 0xB0); /* black on light cyan */
     win_SetAttr  (pcxwin, 0XBB); /* light cyan in window */
     win_SetShadow(pcxwin, 1);
     sed_Repaint(back_sed); /* do it here to avoid time gap */
     sed_Go(back_sed);
     win_Employ(pcxwin); /* Paint win contents and border Title */

     /* close pmap */
     pmap_Close(pmap);
     pmwin_SetPmap(pcxwin, NULL);
  }
  return(pmap != NULL);
} /* display_pcx ***********************************************************/
```

```
/*************** FILE INPUT/OUTPUT VARIABLES HEADER FILE ************/
/* CSCAPE color */
extern int screen_color;

/* USER INTERFACE variables */
extern int units;
extern char crash_forestfile_name[13];


/* FOREST MODEL VARIABLES   ************************************************/
/* GENERAL data file variables; CAPS for consistency with FORTRAN forest model */
extern char
    STSUBM[2],   /* FORTRAN LOGICAL var; treated as 1 char string */
    GMSUBM[2],
    LRAIN[2],
    TEMPC[2];
extern int NYEAR,
    IPYEAR,
    IGYEAR,
    NHOSTS,
    ISYEAR,
    IVIEW[MAX_VIEWS],
    ISEED[6],
    NWEAYR,
    IWFORM,
    IWOPT;
extern float
    TFAC,
    TRETHR;


/* STAND data file variables */
extern char
    METRIC[2],   /* FORTRAN LOGICAL var; treated as 1 char string */
    LDEFOL[2];
extern int ISITE,
    IBOUND,
    ISOPT,
    NSTRYR,
    ICAT;
extern float
    PLOTAR,
    STNDAR,
    HOVER,
    TLIGHT,
    TKL,
    SHX[6][4],    /* misc. variables */
    SHY[6][4],
    STRFAC[2],
    PLOTSZ,
    STRESS[3],
    RSMULT,
    STOKX[6][4],
    STOKY[6][4],
    DLENBASE,
    EPSIGR,
    EPSITR,
    EPSIMG[3],
    PRTMIN,
    DFOLDE,
    STOCKS[3][3],
    CRWDMN,
    SHADMN,
    TEMPMN,
    DFLC[2][2],
    SGMORT;


extern TREE tree[MAX_HOSTS];  /* array of tree structures */


extern MGMT_YRS *start_mgmt_yr_ptr, mgmt_yr_struct;
extern int number_of_mgmt_yrs;         /* number of Management Years entered */


/* DEFOLIATION data file variables used with linked list */
extern DEFOL_YRS *start_defol_yr_ptr, defol_yr_struct;
extern int number_of_defol_yrs;         /* number of Defoliation Years */


/* WEATHER data file variables */
extern float
```

```
    degree_days[MAX_SIM_YRS];  /* total degree days for one year */


/* array of structures */
extern struct ISTR_str  ISTR_array[MAX_SIM_YRS];

/*************************************************************************/
/* DAMAGE MODEL OUTPUT variables */


/* output files data structure */
struct output_file_str {
   char output_needed[2];     /* FORTRAN LOGICAL var; treated as 1 char string */
   boolean SAVE_OUTPUT;       /* TRUE if user wants to save output file */
   char description[31];      /* description of output that is needed */
   FILE *outputfile;          /* file created by damage model */
   char model_outputfile_name[13]; /* name of file created by damage model */
   char user_outputfile_name[13];  /* renamed file created by damage model */
};

/* array of structures */
extern struct output_file_str  outputs[MAX_OUTPUT_FILES];
/*************************************************************************/
/* FILE ACCESS variables */


extern int file_options;

extern FILE *forestfile; /*general, stand, management, weather, species, descriptions*/
/*************************************************************************/
/* FOREST DESCRIPTION variables; defaults loaded 1st;


   21 chars - allow 1 for '\0'*/
extern char *name,
     *job,
     *site,
     *date;
extern char user_notes[MAX_NOTE_SIZE];
extern int line_count;                    /* # of lines in notes file */
/*************************************************************************/
/* GRAPHICS variables */


extern char graph_TNAME[MAX_HOSTS][3];  /* from model */
extern int graph_NHOSTS,    /* from model */
   year_count,              /* number of separate years of output */
   years[MAX_SIM_YRS];      /* array of years for X axis */
extern float Y[MAX_HOSTS][MAX_SIM_YRS];  /* 2D array of amounts for Y axis */
/*************************************************************************/
```

```
/********************** FILE INPUT/OUTPUT FOR FRONT END ***************/
/* Microsoft C 6.0 includes */
#include <string.h>
#include <stdio.h>
/* CSCAPE 3.2 includes */
#include <cscape.h>
/* define file */
#include "defines.h"
/* variables and functions for read/write routines */
#include "struct.h"
#include "fileio.h"


/* FUNCTION PROTOTYPES */
void save_forest (char file_name[13]);
void write_gen(FILE *fptr);
void write_stand (FILE * fptr);
void write_weather(FILE *fptr);
void write_mgmt (FILE * fptr);
void write_one_species (FILE *fptr, int index );
void write_STEMS ( FILE *fptr, int index );
void write_defol (FILE * fptr);
PRIVATE void write_notes( FILE *fptr );
void write_lines_of_notes(FILE *fptr);
void write_ISTR ( FILE *fptr );
void read_forest (char file_name[13]);
PRIVATE void read_gen(FILE *fptr);
PRIVATE void read_stand(FILE *fptr);
PRIVATE void read_weather(FILE *fptr);
PRIVATE void read_mgmt (FILE *fptr);
PRIVATE void read_STEMS ( FILE *fptr,  int index );
PRIVATE void read_defol (FILE *fptr);
PRIVATE void read_notes( FILE *fptr );
PRIVATE void read_ISTR( FILE *fptr );
void read_one_species (FILE *fptr, int index );
void read_XYcoord (FILE *fptr);
void read_graph_NHOSTS (FILE *fptr);
MGMT_YRS *linked_mgmt_yr_insert (MGMT_YRS *list_start, MGMT_YRS new_str );
DEFOL_YRS *linked_defol_yr_insert (DEFOL_YRS *list_start, DEFOL_YRS new_str );


void read_forest (char file_name[13])
/*
DESCRIPTION:
  reads forest file (defaults or user-edited data)
ARGUMENTS:
  file_name is forest file name chosen by the user
SAMPLE CALL:
  read_forest (forestfile_name);
CALLED BY:
  get_file, initialize
CALLS:
  read_gen, read_stand, read_weather, read_mgmt, read_one_species, read_STEMS
  read_defol, read_notes
NOTES:
  file_options flag bits used to determine how to read file data;
*/
{
  int  i;
  forestfile = fopen ( file_name, "rt" );
  if ( forestfile == (FILE *) NULL )
      printf ("%s file could not be opened.\n", file_name );
  fscanf(forestfile, "%d ", &file_options);
  fscanf(forestfile, "%d ", &NHOSTS);
  fscanf(forestfile, "%d\n", &screen_color);
  fscanf(forestfile, "%[^\n]\n", name);
  fscanf(forestfile, "%[^\n]\n", job);
  fscanf(forestfile, "%[^\n]\n", site);
  fscanf(forestfile, "%[^\n]\n", date);

  if ((file_options & GEN) == 1)
     read_gen( forestfile );
  if ((file_options & STAND) == 2)
     read_stand( forestfile );
  for ( i=0; i < NHOSTS; i++ )
  {
     read_one_species ( forestfile, i );
     read_STEMS ( forestfile, i );
  }
  if ((file_options & STDMAN) == 8)
     read_mgmt (forestfile);
  if ((file_options & WEA) == 4)
```

```
      read_weather( forestfile );
   read_defol (forestfile);
   read_ISTR  (forestfile);
   read_notes (forestfile);
   fclose(forestfile);
} /* read_forest ************************************************************/


void read_gen( FILE *fptr )
/*
DESCRIPTION:
  reads gen data from forest file
ARGUMENTS:
  fptr is a pointer to FILE
CALLED BY:
  read_forest
SAMPLE CALL:
  read_gen( forestfile );
*/
{
  int i;

  fscanf(fptr, " %s", STSUBM);
  fscanf(fptr, " %s", GMSUBM);
  fscanf(fptr, " %s", outputs[DEBUG].output_needed);
  fscanf(fptr, "%d", &NYEAR);
  fscanf(fptr, "%d", &ISYEAR);

  fscanf(fptr, "%d", &IPYEAR);
  fscanf(fptr, " %s", outputs[STABLE].output_needed);
  fscanf(fptr, "%d", &IGYEAR);
  fscanf(fptr, " %s", outputs[TGSTEM].output_needed);
  fscanf(fptr, " %s", outputs[TGBA].output_needed);
  fscanf(fptr, " %s", outputs[TGVOL].output_needed);
  fscanf(fptr, " %s", outputs[TGDBH].output_needed);

  for ( i=0; i < MAX_VIEWS; i++ )
     fscanf(fptr, "%d", &IVIEW[i]);
  for ( i=0; i < 6; i++ )
     fscanf(fptr, "%d", &ISEED[i]);
  fscanf(fptr, "%f", &TFAC);
  fscanf(fptr, "%d", &NWEAYR);
  fscanf(fptr, "%f", &TRETHR);
  fscanf(fptr, "%d", &IWFORM);
  fscanf(fptr, "%d", &IWOPT);
  fscanf(fptr, " %s", TEMPC);
  fscanf(fptr, " %s\n", LRAIN);
} /* read_gen ***************************************************************/


void read_stand( FILE *fptr )
/*
DESCRIPTION:
  reads stand data from forest file
ARGUMENTS:
  fptr is a pointer to FILE
CALLED BY:
  read_forest
SAMPLE CALL:
  read_stand( forestfile );
*/
{
  int i, j;
  char  buffer[150];

  fscanf(fptr, "%[^\n]", buffer);
  fscanf(fptr, "%d", &ISITE);
  fscanf(fptr, " %s", METRIC);
  if ((strcmp(METRIC, "T")==0) || (strcmp(METRIC, "t")==0))
     units = SI;
  else
     units = ENGLISH;
  fscanf(fptr, " %s", LDEFOL);
  fscanf(fptr, "%f", &PLOTAR);
  fscanf(fptr, "%f", &STNDAR);
  fscanf(fptr, "%d", &IBOUND);
  fscanf(fptr, "%f\n", &HOVER);
  /* misc. variables */
  fscanf (fptr, "%[^\n]", buffer);
  fscanf(fptr, "%f", &TLIGHT);
  fscanf(fptr, "%f", &TKL);
  for ( j=0; j < 4; j++ )
```

```
      for ( i=0; i < 6; i++ )
         fscanf(fptr, "%f", &SHX[i][j]);
   for ( j=0; j < 4; j++ )
      for ( i=0; i < 6; i++ )
         fscanf(fptr, "%f", &SHY[i][j]);
   fscanf(fptr, "%f", &STRFAC[0]);
   fscanf(fptr, "%f", &STRFAC[1]);
   fscanf(fptr, "%f", &PLOTSZ);
   fscanf(fptr, "%d", &ISOPT);
   for ( i=0; i < 3; i++ )
      fscanf(fptr, "%f", &STRESS[i]);  /* ONLY 3 STRESS VALUES */
   fscanf(fptr, "%f", &RSMULT);
   for ( j=0; j < 4; j++ )
      for ( i=0; i < 6; i++ )
         fscanf(fptr, "%f", &STOKX[i][j]);
   for ( j=0; j < 4; j++ )
      for ( i=0; i < 6; i++ )
         fscanf(fptr, "%f", &STOKY[i][j]);
   fscanf(fptr, "\n");
   for ( i=0; i < 3; i++ )
   {
      for ( j=0; j < 3; j++ )
        fscanf(fptr, "%f", &STOCKS[i][j]);
      fscanf(fptr, "\n");
   }
   fscanf(fptr, "%f", &DFOLDE);
   fscanf(fptr, "%f", &CRWDMN);
   fscanf(fptr, "%f", &SHADMN);
   fscanf(fptr, "%f", &TEMPMN);
   fscanf(fptr, "%f\n", &SGMORT);
   for ( i=0; i < 2; i++ )
      for ( j=0; j < 2; j++ )
         fscanf(fptr, "%f", &DFLC[i][j]);
   fscanf(fptr, "\n");
   fscanf(fptr, "%d", &NSTRYR);
   fscanf(fptr, "%f", &DLENBASE);
   fscanf(fptr, "%d\n", &ICAT);
   fscanf(fptr, "%f", &EPSIGR);
   fscanf(fptr, "%f", &EPSITR);
   for ( i=0; i < 3; i++ )
      fscanf(fptr, "%f", &EPSIMG[i]);
   fscanf(fptr, "%f\n", &PRTMIN);
} /* read_stand ***********************************************************/


void read_weather( FILE *fptr )
/*
DESCRIPTION:
  reads yearly degree days from file
ARGUMENTS:
  fptr is a pointer to FILE
CALLED BY:
  read_forest
SAMPLE CALL:
  read_weather( forestfile );
*/
{
   int i;

   for (i = 0; i < NYEAR; i++)
      fscanf(fptr, "%f\n", &degree_days[i]);
} /* read_weather *********************************************************/


void read_mgmt (FILE *fptr)
/*
DESCRIPTION:
  reads stand management data from forest file and inserts into linked list
ARGUMENTS:
  fptr is a pointer to FILE
CALLED BY:
  read_forest
CALLS:
  linked_mgmt_yr_insert
SAMPLE CALL:
  read_mgmt (forestfile);
*/
{
  int i, j;
  char  buffer[150];

  fscanf(fptr, "%[^\n]", buffer);
```

```
   fscanf(fptr, "%d\n", &number_of_mgmt_yrs);
   for ( i=0; i < number_of_mgmt_yrs; i++ )
   {
      fscanf(fptr, "%d", &mgmt_yr_struct.mgmt_yr_key);
      fscanf(fptr, "%d", &mgmt_yr_struct.manage);
      for ( j=0; j < NHOSTS; j++ )
      {
         fscanf(fptr, "%f",   &mgmt_yr_struct.cutmin[j]);
         fscanf(fptr, "%f",   &mgmt_yr_struct.cutmax[j]);
         fscanf(fptr, "%f\n", &mgmt_yr_struct.action[j]);
      }
      start_mgmt_yr_ptr = linked_mgmt_yr_insert (start_mgmt_yr_ptr, mgmt_yr_struct );
   }
} /* read_mgmt ******************************************************/


void read_one_species ( FILE *fptr,  int i )
/*
DESCRIPTION:
  reads host species data from forest file
ARGUMENTS:
  fptr is a pointer to FILE; i is the host species index (0-5)
CALLED BY:
  read_forest, open_species_file
SAMPLE CALL:
  read_one_species( forestfile, i );
*/
{
  int j, k;

  fscanf(fptr, "%s", tree[i].TNAME);
  fscanf(fptr, "%d", &tree[i].host_code);
  fscanf(fptr, "%d", &tree[i].ISHADE[0]);
  fscanf(fptr, "%d", &tree[i].ISHADE[1]);
  fscanf(fptr, "%f", &tree[i].RESTIN[0]);
  fscanf(fptr, "%f", &tree[i].RESTIN[1]);
  fscanf(fptr, "%f", &tree[i].GROWR);
  fscanf(fptr, "%f", &tree[i].COLD);
  fscanf(fptr, "%f", &tree[i].HOT);
  fscanf(fptr, "%f", &tree[i].SLOWD);
  fscanf(fptr, "%f", &tree[i].CR1);
  fscanf(fptr, "%f", &tree[i].CR2);
  fscanf(fptr, "%f", &tree[i].CR3);
  fscanf(fptr, "%f", &tree[i].CR4);
  fscanf(fptr, "%f", &tree[i].DMAX);
  fscanf(fptr, "%f", &tree[i].HMAX);
  fscanf(fptr, "%f", &tree[i].AGEMAX);
  fscanf(fptr, "%f", &tree[i].F1);
  fscanf(fptr, "%f", &tree[i].F2);
  fscanf(fptr, "%f", &tree[i].SURFAR);
  fscanf(fptr, "%d", &tree[i].ISTOKG);
  fscanf(fptr, "%f", &tree[i].RECRUT);
  for ( k=0; k < 3; k++ )
     for ( j=0; j < 3; j++ )
        fscanf(fptr, "%f", &tree[i].FDIE[j][k]);
  for ( k=0; k < 2; k++ )
     for ( j=0; j < 2; j++ )
        fscanf(fptr, "%d", &tree[i].IDHIST[j][k]);
  fscanf(fptr, "\n");
} /* read_one_species ***********************************************/


void read_STEMS ( FILE *fptr,  int index )
/*
DESCRIPTION:
  reads STEMS data from forest file
ARGUMENTS:
  fptr is a pointer to FILE; index is the host species index (0-5)
CALLED BY:
  read_forest
SAMPLE CALL:
  read_STEMS ( forestfile, i );
*/
{
  int i;

  for ( i=0; i < 20; i++ )
     fscanf(fptr, "%f", &tree[index].STEMS[i]);
  fscanf(fptr, "\n");
} /* read_STEMS *****************************************************/
```

```
void read_defol (FILE *fptr)
/*
DESCRIPTION:
  reads defoliation data from forest file and inserts into linked list
ARGUMENTS:
  fptr is a pointer to FILE
CALLED BY:
  read_forest
CALLS:
  linked_defol_yr_insert
SAMPLE CALL:
  read_defol (forestfile);
*/
{
  int i, j, k;
  char buffer[150];

  fscanf(fptr, "%[^\n]", buffer);
  fscanf(fptr, "%d\n", &number_of_defol_yrs);
  for ( i=0; i < number_of_defol_yrs; i++ )
  {
     fscanf(fptr, "%d", &defol_yr_struct.defol_yr_key);
     for ( k=0; k < 2; k++ )
     {
        for ( j=0; j < NHOSTS; j++ )
           fscanf(fptr, "%d", &defol_yr_struct.defol_amt[k][j]);
        fscanf(fptr, "\n");
     }
     start_defol_yr_ptr = linked_defol_yr_insert (start_defol_yr_ptr, defol_yr_struct );
  }
} /* read_defol ********************************************************/


void read_ISTR( FILE *fptr )
/*
DESCRIPTION:
  initializes ISTR array, reads ISTR array from file
ARGUMENTS:
  fptr is a pointer to FILE
CALLED BY:
  read_forest
SAMPLE CALL:
  read_ISTR( forestfile );
NOTES:
  only reads if NSTRYR > 0
*/
{
   int temp_ISTR;     /* stress year to read */
   int i;

   /* initialize ISTR */
   for (i = 0; i < NYEAR; i++)
   {
      ISTR_array[i].SELECTED = FALSE;
      ISTR_array[i].ISTR = ISYEAR + i;
   }

   /* only reads if NSTRYR > 0 */
   for (i = 0; i < NSTRYR; i++)
   {
      fscanf(fptr, "%d\n", &temp_ISTR);
      /* set the appropriate indexed array element to TRUE */
      ISTR_array[temp_ISTR - ISYEAR].SELECTED = TRUE;
   }

   /* clean up the end of array */
   for (i = NYEAR; i < MAX_SIM_YRS; i++)
   {
       ISTR_array[i].SELECTED = FALSE;
       ISTR_array[i].ISTR = 0;
   }
} /* read_ISTR *********************************************************/


void read_notes( FILE *fptr )
/*
DESCRIPTION:
  reads
ARGUMENTS:
  fptr is a pointer to FILE
CALLED BY:
  read_forest
```

```
SAMPLE CALL:
  read_notes( forestfile );
*/
{
   int chr;    /* character to read */
   int i = 0;

   while (((chr = getc(fptr) ) != EOF) && (i < (MAX_NOTE_SIZE - 2)))
   {
      user_notes[i] = (char)chr;
      i = i+1;
   }
   user_notes[i] = '\0';
} /* read_notes ************************************************************/


/*************************************************************************/
/****************       SAVE FOREST ROUTINES     ************************/
/*************************************************************************/
void save_forest (char file_name[13])
/*
DESCRIPTION:
  saves data as forest file
ARGUMENTS:
  file_name is name chosen by the user to be their forest file
SAMPLE CALL:
  save_forest (forestfile_name);
CALLED BY:
  save_file, exit_routine
CALLS:
  write_gen, write_stand, write_weather, write_mgmt, write_one_species,
  write_STEMS, write_defol, write_notes, write_ISTR
NOTES:
  if user has edited gen, stand, forest management, or weather,
  file_options flag bits used to determine whether to save data to file;
*/
{
  int  i;

  forestfile = fopen ( file_name, "wt" );
  if ( forestfile == (FILE *) NULL )
      printf ("%s file could not be opened.\n", file_name );
  fprintf(forestfile, "%d ", file_options);
  fprintf(forestfile, "%d ", NHOSTS);
  fprintf(forestfile, "%d\n", screen_color);
  fprintf(forestfile, "%s\n", name);
  fprintf(forestfile, "%s\n", job );
  fprintf(forestfile, "%s\n", site);
  fprintf(forestfile, "%s\n", date);

  if ((file_options & GEN) == 1)
     write_gen( forestfile );
  if ((file_options & STAND) == 2)
     write_stand( forestfile );
  for ( i=0; i < NHOSTS; i++ )
  {
     write_one_species ( forestfile, i );
     write_STEMS (forestfile, i);
  }
  if ((file_options & STDMAN) == 8)
     write_mgmt( forestfile );
  if ((file_options & WEA) == 4)
     write_weather( forestfile );
  fprintf(forestfile, " %s\n", "DEFOLIATION - linked list");
  fprintf(forestfile, "%2d\n", number_of_defol_yrs);
  write_defol (forestfile);
  write_ISTR (forestfile);
  write_notes (forestfile);
  fclose(forestfile);
} /* save_forest ***********************************************************/


void write_gen( FILE *fptr )
/*
DESCRIPTION:
  saves gen data to forest file
ARGUMENTS:
  fptr is a pointer to FILE
CALLED BY:
  save_forest, create_model_files
SAMPLE CALL:
  write_gen( forestfile );
```

```
NOTES:
  forest file has already been opened by save_forest
*/
{
  int  i;

  fprintf(fptr, " %s", STSUBM);
  fprintf(fptr, " %s", GMSUBM);
  fprintf(fptr, " %s\n", outputs[DEBUG].output_needed);
  fprintf(fptr, " %3d", NYEAR);
  fprintf(fptr, " %4d\n", ISYEAR);
  fprintf(fptr, " %2d", IPYEAR);
  fprintf(fptr, " %s", outputs[STABLE].output_needed);
  fprintf(fptr, " %2d",   IGYEAR);
  fprintf(fptr, " %s", outputs[TGSTEM].output_needed);
  fprintf(fptr, " %s", outputs[TGBA].output_needed);
  fprintf(fptr, " %s", outputs[TGVOL].output_needed);
  fprintf(fptr, " %s\n", outputs[TGDBH].output_needed);

  for ( i=0; i < MAX_VIEWS; i++ )
  {
     fprintf(fptr, " %4d", IVIEW[i]);
     if (((i+1) % 10) == 0)
        fprintf(fptr, "\n");
  }
  for ( i=0; i < 6; i++ )
     fprintf(fptr, " %7d", ISEED[i]);
  fprintf(fptr, "\n");
  fprintf(fptr, " %5.2f", TFAC);
  fprintf(fptr, " %3d", NWEAYR);
  fprintf(fptr, " %6.2f", TRETHR);
  fprintf(fptr, " %d", IWFORM);
  fprintf(fptr, " %d", IWOPT);
  fprintf(fptr, " %s", TEMPC);
  fprintf(fptr, " %s\n", LRAIN);
} /* write_gen ***********************************************************/


void write_stand ( FILE * fptr )
/*
DESCRIPTION:
  saves stand data to forest file
ARGUMENTS:
  fptr is a pointer to FILE
CALLED BY:
  save_forest, create_model_files
SAMPLE CALL:
  write_stand( forestfile );
*/
{
  int i, j;

  fprintf(fptr, "%s\n", "SITE INFO");
  fprintf(fptr, " %d", ISITE);
  fprintf(fptr, " %s", METRIC);
  fprintf(fptr, " %s\n", LDEFOL);
  fprintf(fptr, " %6.2f", PLOTAR);
  fprintf(fptr, " %6.2f", STNDAR);
  fprintf(fptr, " %d", IBOUND);
  fprintf(fptr, " %7.1f\n", HOVER);
  fprintf (fptr, "%s\n", "MISC. STAND VARIABLES");
  fprintf(fptr, " %4.1f", TLIGHT);
  fprintf(fptr, " %7.5f\n", TKL);
  for ( j=0; j < 4; j++ )
  {
     for ( i=0; i < 6; i++ )
       fprintf(fptr, " %6.2f", SHX[i][j]);
     fprintf(fptr, "\n");
  }
  for ( j=0; j < 4; j++ )
  {
     for ( i=0; i < 6; i++ )
       fprintf(fptr, " %6.2f", SHY[i][j]);
     fprintf(fptr, "\n");
  }
  fprintf(fptr, " %5.2f", STRFAC[0]);
  fprintf(fptr, " %5.2f", STRFAC[1]);
  fprintf(fptr, " %7.4f\n", PLOTSZ);

  fprintf(fptr, " %2d", ISOPT);
  for ( i=0; i < 3; i++ )
     fprintf(fptr, " %4.1f", STRESS[i]);  /* ONLY 3 STRESS VALUES */
```

```
   fprintf(fptr, "\n");
   fprintf(fptr, " %7.4f\n", RSMULT);
   for ( j=0; j < 4; j++ )
   {
      for ( i=0; i < 6; i++ )
        fprintf(fptr, " %4.0f", STOKX[i][j]);
      fprintf(fptr, "\n");
   }
   for ( j=0; j < 4; j++ )
   {
      for ( i=0; i < 6; i++ )
        fprintf(fptr, " %5.2f", STOKY[i][j]);
      fprintf(fptr, "\n");
   }
   for ( i=0; i < 3; i++ )
   {
      for ( j=0; j < 3; j++ )
        fprintf(fptr, " %10.7f", STOCKS[i][j]);
      fprintf(fptr, "\n");
   }
   fprintf(fptr, " %4.2f", DFOLDE);
   fprintf(fptr, " %4.2f", CRWDMN);
   fprintf(fptr, " %4.2f", SHADMN);
   fprintf(fptr, " %4.2f", TEMPMN);
   fprintf(fptr, " %6.3f\n", SGMORT);
   for ( i=0; i < 2; i++ )
      for ( j=0; j < 2; j++ )
         fprintf(fptr, " %11.8f", DFLC[i][j]);
   fprintf(fptr, "\n");
   fprintf(fptr, " %3d",   NSTRYR);
   fprintf(fptr, " %6.2f", DLENBASE);
   fprintf(fptr, " %d\n",  ICAT);
   fprintf(fptr, " %8.6f", EPSIGR);
   fprintf(fptr, " %8.6f", EPSITR);
   for ( i=0; i < 3; i++ )
      fprintf(fptr, " %8.6f", EPSIMG[i]);
   fprintf(fptr, " %8.6f\n", PRTMIN);
} /* write_stand ************************************************************/


void write_weather( FILE *fptr )
/*
DESCRIPTION:
  saves yearly degree days to forest file
ARGUMENTS:
  fptr is a pointer to FILE
CALLED BY:
  save_forest, create_model_files
SAMPLE CALL:
  write_weather( forestfile );
*/
{
   int i;

   for (i = 0; i < NYEAR; i++)
      fprintf(fptr, " %6.1f\n", degree_days[i]);
} /* write_weather ***********************************************************/


void write_mgmt (FILE * fptr)
/*
DESCRIPTION:
  saves forest management years data to forest file
ARGUMENTS:
  fptr is a pointer to FILE
CALLED BY:
  save_forest, create_model_files
SAMPLE CALL:
  write_mgmt( forestfile );
NOTES:
  prints out management years linked list
*/
{
  int j;
  MGMT_YRS *search_ptr;

  fprintf(fptr, " %s\n", "STAND MANAGEMENT PARAMETERS - linked list");
  fprintf(fptr, "%2d\n", number_of_mgmt_yrs);
  search_ptr = start_mgmt_yr_ptr;
  while (search_ptr != NULL_MGMT_PTR)
  {
     fprintf(fptr, " %4d",  search_ptr->mgmt_yr_key);
```

```
      fprintf(fptr, " %d\n", search_ptr->manage);
      for ( j=0; j < NHOSTS; j++ )
      {
         fprintf(fptr, " %6.1f",   search_ptr->cutmin[j]);
         fprintf(fptr, " %6.1f",   search_ptr->cutmax[j]);
         fprintf(fptr, " %6.2f\n", search_ptr->action[j]);
      }
      search_ptr = search_ptr->next;
   }
} /* write_mgmt ************************************************************/


void write_one_species ( FILE *fptr, int i )
/*
DESCRIPTION:
  writes host species data to forest file
ARGUMENTS:
  fptr is a pointer to FILE; i is the host species index (0-5)
CALLED BY:
  save_forest, create_model_files
SAMPLE CALL:
  write_one_species( forestfile, i );
*/
{
  int j, k;

  fprintf(fptr, "%s", tree[i].TNAME);
  fprintf(fptr, "  %3d\n", tree[i].host_code);
  fprintf(fptr, " %2d", tree[i].ISHADE[0]);
  fprintf(fptr, " %2d", tree[i].ISHADE[1]);
  fprintf(fptr, " %8.4f", tree[i].RESTIN[0]);
  fprintf(fptr, " %8.4f", tree[i].RESTIN[1]);
  fprintf(fptr, " %7.1f", tree[i].GROWR);
  fprintf(fptr, " %6.0f", tree[i].COLD);
  fprintf(fptr, " %6.0f", tree[i].HOT);
  fprintf(fptr, " %6.3f\n", tree[i].SLOWD);
  fprintf(fptr, " %6.2f", tree[i].CR1);
  fprintf(fptr, " %7.4f", tree[i].CR2);
  fprintf(fptr, " %6.2f", tree[i].CR3);
  fprintf(fptr, " %6.3f\n", tree[i].CR4);
  fprintf(fptr, " %7.1f", tree[i].DMAX);
  fprintf(fptr, " %7.0f", tree[i].HMAX);
  fprintf(fptr, " %4.0f", tree[i].AGEMAX);
  fprintf(fptr, " %8.5f", tree[i].F1);
  fprintf(fptr, " %8.5f", tree[i].F2);
  fprintf(fptr, " %6.1f", tree[i].SURFAR);
  fprintf(fptr, " %2d", tree[i].ISTOKG);
  fprintf(fptr, " %6.0f\n", tree[i].RECRUT);
  for ( k=0; k < 3; k++ )
     for ( j=0; j < 3; j++ )
        fprintf(fptr, " %4.2f", tree[i].FDIE[j][k]);
  fprintf(fptr, "\n");
  for ( k=0; k < 2; k++ )
     for ( j=0; j < 2; j++ )
        fprintf(fptr, " %4d", tree[i].IDHIST[j][k]);
  fprintf(fptr, "\n");
} /* write_one_species *****************************************************/


void write_STEMS ( FILE *fptr, int index )
/*
DESCRIPTION:
  writes STEMS data to forest file
ARGUMENTS:
  fptr is a pointer to FILE; index is the host species index (0-5)
CALLED BY:
  save_forest, create_model_files
SAMPLE CALL:
  write_STEMS ( forestfile, i );
*/
{
  int i;

  for ( i=0; i < 20; i++ )
  {
     fprintf(fptr, " %6.1f", tree[index].STEMS[i]);
     if (((i+1) % 10) == 0)
        fprintf(fptr, "\n");
  }
} /* write_STEMS */
```

```
void write_defol (FILE * fptr)
/*
DESCRIPTION:
  saves forest defoliation years data to forest file
ARGUMENTS:
  fptr is a pointer to FILE
CALLED BY:
  save_forest, create_model_files
SAMPLE CALL:
  write_defol( forestfile );
NOTES:
  defoliation years linked list
*/
{
  int j, k;
  DEFOL_YRS *search_ptr;

  search_ptr = start_defol_yr_ptr;
  while (search_ptr != NULL_DEFOL_PTR)
  {
     fprintf(fptr, " %4d\n",  search_ptr->defol_yr_key);
     for ( k=0; k < 2; k++ )
     {
        for ( j=0; j < NHOSTS; j++ )
           fprintf(fptr, " %3d", search_ptr->defol_amt[k][j]);
        fprintf(fptr, "\n");
     }
     search_ptr = search_ptr->next;
  }
} /* write_defol ************************************************************/


void write_ISTR ( FILE *fptr )
/*
DESCRIPTION:
  writes stress data to forest file
ARGUMENTS:
  fptr is a pointer to FILE
CALLED BY:
  save_forest, create_model_files
SAMPLE CALL:
  write_ISTR ( forestfile );
*/
{
  int i;

  for (i = 0; i < NYEAR; i++)
  {
     if (ISTR_array[i].SELECTED == TRUE)
        fprintf(fptr, " %4d\n", ISTR_array[i].ISTR);
  }
} /* write_ISTR */


void write_notes( FILE *fptr )
/*
DESCRIPTION:
  reads
ARGUMENTS:
  fptr is a pointer to FILE
CALLED BY:
  save_forest
SAMPLE CALL:
  write_notes( forestfile );
*/
{
   fprintf(fptr, "%s\0", user_notes);
} /* write_notes ************************************************************/


void write_lines_of_notes(FILE *fptr)
/*
DESCRIPTION:
  writes notes in separate lines so FORTRAN model can print in table;
  does not split words at end of lines
ARGUMENTS:
  fptr is a pointer to FILE
CALLED BY:
  create_model_files
SAMPLE CALL:
  write_lines_of_notes (notesfile);
*/
```

```
{
   int NEWLINE = FALSE;
   int j;
   int k = 0;
   int chr_pos = 0;     /* character position within user_notes */
   int length;          /* length of user_notes */
   int blank_position = LINE_SIZE;  /* initially, no blanks; 0 -> start of line*/
   char line_buf1[LINE_SIZE+1];     /* buffers to hold strings built from user_notes */
   char line_buf2[LINE_SIZE+1];

   length = strlen (user_notes);
   if (length > 0)
   {
      while ((chr_pos + LINE_SIZE) < length)  /* treat last line differently */
      {
         /* first, find line feed char '\n' (ASCII 10)*/
         NEWLINE = FALSE;
         for (j = 0; j < LINE_SIZE+1; j++)
            if ((int)user_notes[chr_pos+j] == 10)  /* find line feed '\n' */
               NEWLINE = TRUE;

         /* if newline, write line, increment chr_pos */
         if (TRUE == NEWLINE)
         {
            /* build line up to newline */
            while ((int)user_notes[chr_pos + k] != 10)
            {
               line_buf2[k] = user_notes[chr_pos + k];
               k = k+1;
            }
            line_buf2[k] = '\0';
            fprintf(fptr, "%s\n", line_buf2);
            line_count = line_count + 1;
            chr_pos = chr_pos + k + 1;
            k = 0;
         }
         /* if beginning of next line is a blank (ASCII 32), write line as is */
         else if ((int)user_notes[chr_pos+LINE_SIZE] == 32)
         {
            /* build line, write line, increment chr_pos */
            for (j = 0; j < LINE_SIZE; j++)
               line_buf2[j] = user_notes[chr_pos+j];
            line_buf2[LINE_SIZE] = '\0';
            fprintf(fptr, "%s\n", line_buf2);
            line_count = line_count + 1;
            chr_pos = chr_pos + LINE_SIZE +1;
         }
         else
         {
            /* build line, find blank location */
            for (j = 0; j < LINE_SIZE; j++)
            {
               line_buf1[j] = user_notes[chr_pos+j];
               if ((int)line_buf1[j] == 32)          /* char = BLANK */
                  blank_position = j;
            }
            /* copy piece to line 2, write it */
            strncpy (line_buf2, line_buf1, blank_position);
            line_buf2[blank_position] = '\0';
            fprintf(fptr, "%s\n", line_buf2);
            line_count = line_count + 1;
            chr_pos = chr_pos + blank_position + 1;
            /* reset blank_position and buffer */
            blank_position = LINE_SIZE;
            strcpy (line_buf1, "\0");
         }
      } /* end while */

      /* build last line, copy piece to line 2, write it */
      for (j = 0; j < (length - chr_pos); j++)
      {
         line_buf1[j] = user_notes[chr_pos+j];
         /* if line feed is in buffer, add to line count */
         if ((int)line_buf1[j] == 10)
            line_count = line_count + 1;
      }
      strncpy (line_buf2, line_buf1, j);
      line_buf2[j] = '\0';
      fprintf(fptr, "%s\n", line_buf2);
      line_count = line_count + 1;
   } /* end if */
} /* write_lines_of_notes ************************************************/
```

```
void read_graph_NHOSTS (FILE * fptr)
/*
DESCRIPTION:
  model generated values
CALLED BY:
  control_output_draw
*/
{
  int i;

  fscanf(fptr, "%d\n", &graph_NHOSTS);
  for ( i = 0; i < graph_NHOSTS; i++ )
     fscanf(fptr, "%s", graph_TNAME[i]);
  fclose(fptr);
} /* read_graph_NHOSTS*******************************************************/


void read_XYcoord (FILE * fptr)
/*
DESCRIPTION:
  reads table file
CALLED BY:
  control_output_draw
*/
{
  int  i = 0;
  int j;

  while (!feof (fptr))  /* read until all years read */
  {
     fscanf(fptr, "%d", &years[i]);
     for ( j = 0; j < graph_NHOSTS; j++ )
         fscanf(fptr, "%f", &Y[j][i]);
     fscanf(fptr, "\n");
     if (years[i] > 0)       /* "while" continues when blank line in file */
         year_count++;        /* MUST NOT add any more years */
     i++;
   }
   fclose(fptr);
} /* read_XYcoord************************************************************/
```

```
/**************** STAND VARIABLES HEADER FILE ************/
/* CSCAPE window variables */
extern int win_width,      /* window width */
    win_x,          /* window position (x coord) of upper left hand corner */
    win_y,          /* window position (y coord) of upper left hand corner */
    win_height;     /* window height */

extern boolean CHOICE;


/* GENERAL data file variables; CAPS for consistency with FORTRAN forest model */
extern int NYEAR,
    ISYEAR,
    ISEED[6];
extern float TFAC;

extern MGMT_YRS *start_mgmt_yr_ptr;
extern DEFOL_YRS *start_defol_yr_ptr;

/* STAND data file variables */
extern char
    METRIC[2];
extern int ISITE,
    IBOUND,
    ISOPT,
    NSTRYR;
extern float
    PLOTAR,
    STNDAR,
    HOVER,
    TLIGHT,
    TKL,
    SHX[6][4],    /* misc. variables */
    SHY[6][4],
    STRFAC[2],
    PLOTSZ,
    STRESS[3],
    RSMULT,
    STOKX[6][4],
    STOKY[6][4],
    DLENBASE,
    EPSIGR,
    EPSITR,
    EPSIMG[3],
    PRTMIN,
    DFOLDE,
    STOCKS[3][3],
    CRWDMN,
    SHADMN,
    TEMPMN,
    DFLC[2][2],
    SGMORT;

/* array of structures */
extern struct ISTR_str  ISTR_array[MAX_SIM_YRS];

/* FILE ACCESS variables */
extern int file_options;

/* USER INTERFACE variables */
extern boolean EDITS_MADE;
extern int units;
```

```
/********************** STAND EDIT ***********************************/
/* Microsoft C 6.0 includes */
#include <stdio.h>
/* CSCAPE 3.2 includes */
#include <cscape.h>
#include <popdecl.h>
#include "defines.h"
#include "struct.h"
#include "stand.h"

#define FT_to_M                0.3048


/* FUNCTION PROTOTYPES */
/* routines using CSCAPE function calls */
int stand_edit(VOID *sdata, int idata);
int stand_data(VOID *sdata, int idata);
PRIVATE int choose_ISTR_yrs (void);
int gen_edit (VOID *sdata, int idata);

/* no CSCAPE */
PRIVATE void do_years_agree (void);
PRIVATE void adjust_ISTR_array (int initial_NYEAR);
void mem_check(boolean returned);
int getkey_stroke(void);


int gen_edit(VOID *sdata, int idata)
/*
DESCRIPTION:
  edit general data
CALLED BY:
  framer menu GENERAL DATA
CALLS:
  do_years_agree; adjust_ISTR_array
*/
{
   menu_type menu;
   sed_type gen_sed;
   int initial_ISYEAR,         /* used to check for change in value */
       initial_NYEAR;
   char units_string[8];

   EDITS_MADE = TRUE;
   file_options = file_options | GEN;
   initial_ISYEAR = ISYEAR;
   initial_NYEAR  = NYEAR;
   if ((menu = menu_Open()) == NULL)
   {
      printf ("unable to open menu");
      return(0);
   }
   win_width = 61;
   win_y = 11;
   win_x = 10;
   if ((strcmp(METRIC, "T")==0) || (strcmp(METRIC, "t")==0)) {
      units = SI;
      strcpy (units_string, "METRIC");
   }
   else  {
      units = ENGLISH;
      strcpy (units_string, "ENGLISH");
   }
   menu_Printf(menu, "Starting Year: @fd2[#####]\n",
     &ISYEAR,&int_funcs,"Simulation Starting Year; > 0; ISYEAR", "(1,)");
   menu_Printf(menu, "Years to Simulate: @fd2[####]\n",
     &NYEAR,&int_funcs,"Years to simulate; 1->100; NYEAR", "(1,100)");
   menu_Printf(menu, "Random Number Seed: @fd2[#####]\n",
      &ISEED[0], &int_funcs,"Tree Establishment Seed; > 0; ISEED[1]", "(0,)");
   menu_Printf(menu, "Random Number Seed: @fd2[#####]\n",
      &ISEED[2], &int_funcs,"Tree Stress Mortality; > 0; ISEED[3]", "(0,)");
   menu_Printf(menu,"Heat multiplier: @fd2[######]\n",
      &TFAC,&sfloat_funcs,"Effects accum. annual day-degrees for all yrs;0.1->10.0; TFAC", "(0.1,10)");
   menu_Printf(menu, "Units: @fd2[#######]",
      units_string, &toggle_funcs, "Choose between METRIC units and ENGLISH units with SPACE BAR",
         "METRIC,ENGLISH");
   menu_Flush(menu);
   if ((gen_sed = sed_Open(menu)) == NULL)  {
      printf ("unable to open sed");
      return(0);
   }
   sed_SetLabel(gen_sed, 26);
```

```
    sed_SetWidth(gen_sed, win_width );
    sed_SetPosition(gen_sed, win_y, win_x);
    sed_SetColors (gen_sed, CYAN_BLUE, WHITE_BLUE, YELLOW_CYAN);
    sed_SetShadow(gen_sed, 1);
    sed_SetShadowAttr(gen_sed, DGRAY_BLACK);
    sed_SetMouse(gen_sed, sedmou_GreedyTrack);
    sed_SetBorder(gen_sed, bd_std);
    sed_SetBorderTitle(gen_sed, "    GENERAL DATA");
    sed_Repaint(gen_sed);
    sed_Go(gen_sed);
    if ((initial_ISYEAR != ISYEAR) || (initial_NYEAR != NYEAR))  {
       do_years_agree ();
       adjust_ISTR_array (initial_NYEAR);
    }
    sed_Close(gen_sed);
    if (strcmp(units_string, "METRIC")==0) {
       units = SI;
       strcpy (METRIC, "T");
       /* METRIC part of stand data, default value is F */
       file_options = file_options | STAND;
    }
    else  {
       units = ENGLISH;
       strcpy (METRIC, "F");
    }
    return (0);
} /* gen_edit ******************************************************/


void adjust_ISTR_array (int initial_NYEAR)
/*
DESCRIPTION:
  change ISTR year array
ARGUMENTS:
  initial_NYEAR is NYEAR before editing changed it
CALLED BY:
  gen_edit
*/
{
  int i, j;
  /* array of temporary structures */
  struct ISTR_str  temp_ISTR_array[MAX_SIM_YRS];

  /* initialize temporary array */
  for (i = 0; i < NYEAR; i++)  {
      temp_ISTR_array[i].SELECTED = 0;
      temp_ISTR_array[i].ISTR = ISYEAR + i;
  }

  /*go through years, if SELECTED and years match, set temporary SELECTED array element to TRUE */
  for (j = 0; j < initial_NYEAR; j++)
     for (i = 0; i < NYEAR; i++)
        if ((ISTR_array[j].SELECTED == 1) && (ISTR_array[j].ISTR == temp_ISTR_array[i].ISTR))
           temp_ISTR_array[i].SELECTED = 1;

  /* ISTR_array gets temporary array values */
  for (i = 0; i < NYEAR; i++)  {
      ISTR_array[i].SELECTED = temp_ISTR_array[i].SELECTED;
      ISTR_array[i].ISTR = temp_ISTR_array[i].ISTR;
  }

  /* clean up the end of array */
  for (i = NYEAR; i < MAX_SIM_YRS; i++)  {
      ISTR_array[i].SELECTED = FALSE;
      ISTR_array[i].ISTR = 0;
  }
}  /* adjust_ISTR_array *******************************************/


void do_years_agree (void)
/*
DESCRIPTION:
  check for a change in starting year and # of simulation years that may affect
  the management, defoliation, and weather years
CALLED BY:
  gen_edit
*/
{
  MGMT_YRS *mgmt_search_ptr;
  DEFOL_YRS *defol_search_ptr;
  boolean MGMT_ERROR = FALSE;
  boolean DEFOL_ERROR = FALSE;
```

```
   mgmt_search_ptr = start_mgmt_yr_ptr;
   while (mgmt_search_ptr != NULL_MGMT_PTR)  {
      /* is mgmt year outside of new range of years? */
      if ((mgmt_search_ptr->mgmt_yr_key < ISYEAR) ||
          (mgmt_search_ptr->mgmt_yr_key > (ISYEAR + NYEAR)))
         MGMT_ERROR = TRUE;
      mgmt_search_ptr = mgmt_search_ptr->next;
   }
   if (MGMT_ERROR == TRUE)  {
      beep();
      prompt ("WARNING: Management years are now outside of range of designated simulation years");
   }

   defol_search_ptr = start_defol_yr_ptr;
   while (defol_search_ptr != NULL_DEFOL_PTR)  {
      /* is defol year outside of new range of years? */
      if ((defol_search_ptr->defol_yr_key < ISYEAR) ||
          (defol_search_ptr->defol_yr_key > (ISYEAR + NYEAR)))
         DEFOL_ERROR = TRUE;
      defol_search_ptr = defol_search_ptr->next;
   }
   if (DEFOL_ERROR == TRUE)  {
      beep();
      prompt ("WARNING: Defoliation years are now outside of range of designated simulation years");
   }

   /* has user edited weather data? */
   if ((file_options & WEA) == 4)  {
      beep();
      prompt
      ("WARNING: Altered start year or length of simulation may have desynchronized weather data. Check
weather data");
   }
}  /* do_years_agree ****************************************************/


int stand_edit(VOID *sdata, int idata)
/*
DESCRIPTION:
  edit stand data
CALLED BY:
  framer menu STAND DETAILS, stand_data
CALLS:
  choose_ISTR_yrs
*/
{
  menu_type menu;
  sed_type sed;
  int i, j, k;

  if (units == SI)  {
     PLOTSZ *= AC_to_HA;
     if (IBOUND == 1)  /* height */
        HOVER *= FT_to_M;
     else              /* diameter */
        HOVER *= IN_to_CM;
  }
  EDITS_MADE = TRUE;
  file_options = file_options | STAND;
  yellow_message ("One Moment Please");
  if ((menu = menu_Open()) == NULL)  {
     printf ("unable to open menu");
     return(0);
  }
  win_width = 78;
  win_height = disp_GetHeight()-3;
  win_y = 1;
  win_x = 1;
  k = 1;
  if (IBOUND == 1)  /* height */
     mem_check(menu_Printf(menu,"Boundary Height: @fd2[@[8,#]]\n",
       &HOVER, &sfloat_funcs,"Height  > 0; HOVER", "(0.1,4000)"));
  else  /* diameter */
     mem_check(menu_Printf(menu,"Boundary Diameter: @fd2[@[8,#]]\n",
       &HOVER, &sfloat_funcs,"Diameter; > 0; HOVER", "(0,50)"));
  mem_check(menu_Printf(menu,"Solar insolation factor: @fd2[#####]\n",
    &TLIGHT,&sfloat_funcs,"Average annual insolation conversion in Eq. 10; 0.0 -> 1.0; TLIGHT", "(0,1)"));
  mem_check(menu_Printf(menu,"Shade effect on diameter growth - light degradation rate: @fd2[@[7,#]]\n",
    &TKL,&sfloat_funcs,"Shading surface area rate coefficient in eq. 10; 0.0 -> 1.0; TKL", "(0,1)"));
  mem_check(menu_Printf(menu,"@[76,%c]\n", 205));
  mem_check(menu_Printf(menu,"@a[0x1B]Effect of Available Light on Diameter Growth (X, Y values of
interpolation)\n"));
```

```
  mem_check(menu_Printf(menu,"     by tree shade tolerance class@a[0x17]\n"));
  mem_check(menu_Printf(menu,"\n"));
  mem_check(menu_Printf(menu,"Intolerant   Intolerant  Tolerant    Tolerant    Red Maple    Red Oak\n" ));
  mem_check(menu_Printf(menu,"             /Intermed.  /Intermed.\n"));
  mem_check(menu_Printf(menu,"@[76,%c]\n", 196));
  mem_check(menu_Printf(menu,"   X    Y      X    Y      X    Y      X    Y      X    Y      X    Y%s","\n"));
  mem_check(menu_Printf(menu,"@[76,%c]\n", 196)); /*ASCII 196 - straight line*/
  for ( j=0; j < 4; j++ )
     for ( i=0; i < 6; i++ )  {
       mem_check(menu_Printf(menu,"   @fd2[####]",
         &SHX[i][j],&sfloat_funcs,"Available light, ALIGHT (X coordinate); 0.0 -> 1.0; SHX", "(0,1)"));
       mem_check(menu_Printf(menu," @fd2[####]",
         &SHY[i][j],&sfloat_funcs,"Shading factor SHADE    (Y coordinate); 0.0 -> 1.0; SHY", "(0,1)"));
       if (((i+1) % 6) == 0)
          mem_check(menu_Printf(menu, "\n"));
     }
  mem_check(menu_Printf(menu,"@[76,%c]\n", 205));
  mem_check(menu_Printf(menu,"Shading Influence Area: @fd2[@[7,#]]\n",
    &PLOTSZ, &sfloat_funcs,"Area to calculate influence of surrounding trees; > 0; PLOTSZ", "(0.0001,)"));
  mem_check(menu_Printf(menu,"Minimum value for Shade: @fd2[#####]\n",
    &SHADMN, &sfloat_funcs,"Minimum value for SHADE effect on diameter growth; 0.0 -> 1.0; SHADMN", "(0,1)"));
  mem_check(menu_Printf(menu,"Minimum value for Temperature effect: @fd2[#####]\n",
    &TEMPMN, &sfloat_funcs,"Minimum value for temperature effect on diameter growth; 0.0 -> 1.0; TEMPMN",
"(0,1)"));
  mem_check(menu_Printf(menu,"Minimum value for effect of CROWDing on diameter growth: @fd2[#####]\n",
    &CRWDMN, &sfloat_funcs,"Minimum value for use of CROWD (eqn. 12) in eqn. 16; 0.0 -> 1.0; CRWDMN", "(0,1)"));
  mem_check(menu_Printf(menu,"Slope of defoliation effect on foliage production: @fd2[#####]\n",
    &DFOLDE, &sfloat_funcs,"Slope of line; 0.0 -> 1.0; DFOLDE", "(0,1)"));
  mem_check(menu_Printf(menu,"@[76,%c]\n", 205));
  mem_check(menu_Printf(menu,"    @a[0x1B]Effect of Defoliation on Diameter Growth@a[0x17]\n"));
  mem_check(menu_Printf(menu,"Intercept for effect of defoliation:  @fd2[@[11,#]]\n",
    &DFLC[0][0],&sfloat_funcs,"Y-intercept for eqn. 13: 0.8 -> 1.0; DFLC(1,1)", "(0.8,1)"));
  mem_check(menu_Printf(menu,"Slope for the effect of defoliation:  @fd2[@[11,#]]\n",
    &DFLC[0][1],&sfloat_funcs,"Slope for eqn. 13: -0.008 -> 0.0; DFLC(1,2)", "(-0.008,0)"));
  mem_check(menu_Printf(menu,"Coefficient for effect of defoliation:  @fd2[@[11,#]]\n",
    &DFLC[1][0],&sfloat_funcs,"Denominator, first term in eqn. 14: 1.0 -> 5.0; DFLC(2,1)", "(1,5)"));
  mem_check(menu_Printf(menu,"Intercept for effect of defoliation:  @fd2[@[11,#]]\n",
    &DFLC[1][1],&sfloat_funcs,"Denominator, defoliation multiplier in eqn. 14: 0.0 -> 1.0; DFLC(2,2)",
"(0,1)"));
  mem_check(menu_Printf(menu,"@[76,%c]\n", 205));
  mem_check(menu_Printf(menu,"    @a[0x1B]Quadratic equation coefficients to produce relative stocking
RSTOCK\n"));
  mem_check(menu_Printf(menu,"      by stocking class x@a[0x17]\n"));
  mem_check(menu_Printf(menu,"     STOCKS(x,1)   STOCKS(x,2)   STOCKS(x,3)\n" ));
  mem_check(menu_Printf(menu,"@[76,%c]\n", 196));
  for ( i=0; i < 3; i++ )  {
    mem_check(menu_Printf(menu, "x=%d: ",i+1));
    mem_check(menu_Printf(menu,"@fd2[@[11,#]]",
      &STOCKS[i][0],&sfloat_funcs,"Constant coefficient for relative stocking RSTOCK;-0.1 -> 0.1; STOCKS(x,1)",
"(-0.1,0.1)"));
    mem_check(menu_Printf(menu,"   @fd2[@[11,#]]",
      &STOCKS[i][1],&sfloat_funcs,"Linear coefficient for relative stocking RSTOCK;0 -> 0.1; STOCKS(x,2)",
"(0,0.1)"));
    mem_check(menu_Printf(menu,"   @fd2[@[11,#]]\n",
      &STOCKS[i][2],&sfloat_funcs,"Quadratic coefficient for relative stocking RSTOCK;0 -> 0.05; STOCKS(x,3)",
"(0,0.05)"));
  }
  mem_check(menu_Printf(menu,"@[76,%c]\n", 205));
  mem_check(menu_Printf(menu,"Relative stocking effect: @fd2[######]\n",
    &RSMULT, &sfloat_funcs,"Relative stocking to tree crowding slope; 0.0 -> 1.0; RSMULT", "(0,)"));
  mem_check(menu_Printf(menu,"@[76,%c]\n", 205));
  mem_check(menu_Printf(menu,"   @a[0x1B]Effect of relative stocking on recruitment (values for
interpolation)@a[0x17]\n"));
  mem_check(menu_Printf(menu,"\n"));
  mem_check(menu_Printf(menu,"Intolerant   Intolerant  Tolerant    Tolerant    Red Maple    Not Used\n" ));
  mem_check(menu_Printf(menu,"             /Intermed.  /Intermed.\n"));
  mem_check(menu_Printf(menu,"@[76,%c]\n", 196));
  mem_check(menu_Printf(menu,"   X    Y      X    Y      X    Y      X    Y      X    Y      X    Y%s","\n"));
  mem_check(menu_Printf(menu,"@[76,%c]\n", 196));
  for ( j=0; j < 4; j++ )
     for ( i=0; i < 6; i++ )  {
       mem_check(menu_Printf(menu,"   @fd2[####]",
          &STOKX[i][j],&sfloat_funcs,"Relative stocking RSTOCK (X coordinate); 0 -> 120; STOKX", "(0,120)"));
       mem_check(menu_Printf(menu," @fd2[####]",
          &STOKY[i][j],&sfloat_funcs,"Proportion max. trees recruited (RSHADE: Y coordinate); 0.0 -> 1.0;
STOKY", "(0,1)"));
       if (((i+1) % 6) == 0)
          mem_check(menu_Printf(menu,"\n"));
     }
  mem_check(menu_Printf(menu,"@[76,%c]\n", 205));
  mem_check(menu_Printf(menu,"Minimum Stem count sufficient to perform growth calc.: @fd2[#####]\n",
    &EPSIGR, &sfloat_funcs,"Bound for STEM(IH,ID) to compute growth; 0.0 -> 1.0; EPSIGR", "(0,1)"));
```

```
  mem_check(menu_Printf(menu,"Sufficient stem count to carry stems to next size class:  @fd2[#####]\n",
    &EPSITR, &sfloat_funcs,"TREEN lower limit for diameter growth; 0.0 -> 1.0; EPSITR", "(0,1)"));
  mem_check(menu_Printf(menu,"Target thinning STEM count lower limit: Epsilon 1: @fd2[#####]\n",
    &EPSIMG[0], &sfloat_funcs,"STEM(IH,ID) large enough to be considered for removal; 0.0 -> 1.0; EPSIMG[1]",
"(0,1)"));
  mem_check(menu_Printf(menu,"Lower limit for total stem count for removal: Epsilon 2: @fd2[#####]\n",
    &EPSIMG[1], &sfloat_funcs,"TSUMDC: total between CUTMIN & CUTMAX; 0.0 -> 1.0; EPSIMG[2]", "(0,1)"));
  mem_check(menu_Printf(menu,"Lower bound for total stems to be cut: @fd2[#####]\n",
    &EPSIMG[2], &sfloat_funcs,"Stem available needed to warrant action; 0.0 -> 1.0; EPSIMG[3]", "(0,1)"));
  mem_check(menu_Printf(menu,"Minimum count of total stems in a strata to be displayed: @fd2[#####]\n",
    &PRTMIN, &sfloat_funcs,"Lower limit for host-canopy strata to be included in output table;0->1; PRTMIN",
"(0,1)"));
  mem_check(menu_Printf(menu,"Exponential decay rate: @fd2[@[7,#]]\n",
    &SGMORT, &sfloat_funcs,"Years of slow growth on tree mortality rate: -0.5 -> 0.0; SGMORT", "(-0.5,0)"));
  mem_check(menu_Printf(menu,"Non-stress Mortality Factor: @fd2[####]\n",
    &STRFAC[0], &sfloat_funcs,"Value of SDIE when no defoliation stress present: > 0.0; STRFAC[1]", "(0,)"));
  mem_check(menu_Printf(menu,"Stress Induced Mortality Factor: @fd2[####]\n",
    &STRFAC[1], &sfloat_funcs,"Value of SDIE (eqn. 19) when stress present; > 0; STRFAC[2] ", "(0,)"));
  for ( i=0; i < 3; i++ )
    mem_check(menu_Printf(menu,"Stress Probability (Site %d):@fd2[###]\n", k++,
      &STRESS[i],&sfloat_funcs,"Prob. added stress occurs beyond defol. stress: 0.0 -> 1.0; STRESS", "(0,1)"));
  mem_check(menu_Printf(menu, "Stress option: @fd2[###]\n",
    &ISOPT, &int_funcs,"0:no stress, 1:random stress, 2:specify years of stress; 0 -> 2; ISOPT", "(0,2)"));
  menu_Flush(menu);
  if ((sed = sed_Open(menu)) == NULL) {
     printf ("unable to open sed");
     menu_Destroy(menu);
     return(0);
  }
  sed_SetLabel(sed, 28);
  sed_SetWidth(sed, win_width);
  sed_SetHeight(sed, win_height);
  sed_SetPosition(sed, win_y, win_x);
  sed_SetColors (sed, CYAN_BLUE, WHITE_BLUE, YELLOW_CYAN);
  sed_SetShadowAttr(sed, DGRAY_BLACK);
  sed_SetMouse(sed, sedmou_GreedyTrack);
  sed_SetBorder(sed, bd_mouse);
  sed_SetBorderTitle(sed, "STAND  DETAILS");
  sed_SetSpecial(sed, inter_field_grid);
  sed_Repaint(sed);
  sed_Go(sed);
  win_height = 18; /* reset */
  yellow_message (NULL);
  if (ISOPT == 2)
     ISOPT = choose_ISTR_yrs ();
  sed_Close(sed);
  if (units == SI) {
     PLOTSZ /= AC_to_HA;
     if (IBOUND == 1)  /* height */
        HOVER /= FT_to_M;
     else              /* diameter */
        HOVER /= IN_to_CM;
  }
  return (0);
} /* stand_edit ************************************************/


int stand_data(VOID *sdata, int idata)
/*
DESCRIPTION:
  edit stand details
CALLED BY:
  framer menu
*/
{
  menu_type menu;
  sed_type sed;
  float temp_DLENBASE;
  char toggle[30];

  EDITS_MADE = TRUE;
  file_options = file_options | STAND;

  if (units == SI)  {
     PLOTAR *= AC_to_HA;
     STNDAR *= AC_to_HA;
     DLENBASE *= IN_to_CM;
  }
  temp_DLENBASE = DLENBASE;
  if (IBOUND == 1)  /* height */
     strcpy(toggle, "Tree Height");
  else
```

```
      strcpy(toggle, "Tree Diameter");
   if ((menu = menu_Open()) == NULL)  {
      printf ("unable to open menu");
      return(0);
   }
   win_width = 65;
   win_y = 12;
   win_x = 10;
   mem_check(menu_Printf(menu, "Site Moisture Index: @fd2[###]\n",
     &ISITE, &int_funcs,"1 = wet  2 = medium  3 = dry;  ISITE", "(1,3)"));
   mem_check(menu_Printf(menu,"Tree Sample Plot Area: @fd2[@[8,#]]\n",
     &PLOTAR,&sfloat_funcs,"Area used to obtain stem counts by species; 0.01 -> 999.99; PLOTAR",
"(0.009,999.99)"));
   mem_check(menu_Printf(menu,"Stand Area: @fd2[@[8,#]]\n",
     &STNDAR,&sfloat_funcs,"Total area: spatially homogenous; 0.01 -> 999.99; STNDAR", "(0.009,999.99)"));
   mem_check(menu_Printf(menu, "Overstory/Understory Boundary Type: @fd2[@[13,#]]\n",
     toggle, &toggle_funcs,"Space bar toggles between 2 types: Tree Height or Tree Diameter", "Tree Height,Tree
Diameter"));
   mem_check(menu_Printf(menu,"Diameter class width: @fd2[######]\n",
     &DLENBASE, &sfloat_funcs,"Diameter width for all diameter classes; 2.0 -> 4.0; DLENBASE", "(2,10.16)"));
   menu_Flush(menu);
   if ((sed = sed_Open(menu)) == NULL) {
      printf ("unable to open sed");
      menu_Destroy(menu);
      return(0);
   }
   sed_SetLabel(sed, 61);
   sed_SetWidth(sed, win_width);
   sed_SetPosition(sed, win_y, win_x);
   sed_SetColors (sed, CYAN_BLUE, WHITE_BLUE, YELLOW_CYAN);
   sed_SetShadowAttr(sed, DGRAY_BLACK);
   sed_SetMouse(sed, sedmou_GreedyTrack);
   sed_SetBorder(sed, bd_std);
   sed_SetBorderTitle(sed, "              STAND DATA");
   sed_Repaint(sed);
   sed_Go(sed);
   win_height = 18; /* reset */
   if (strcmp(toggle, "Tree Height")==0)
      IBOUND = 1;
   else
      IBOUND = 2;
   if (temp_DLENBASE != DLENBASE)
     prompt
("WARNING: Change made to diameter class width will necessitate changes to stem counts in each class for all
species.");
   sed_Close(sed);
   if (units == SI)  {        /* change back to original units */
      PLOTAR /= AC_to_HA;
      STNDAR /= AC_to_HA;
      DLENBASE /= IN_to_CM;
   }
   return (0);
} /* stand_data ************************************************/


int choose_ISTR_yrs (void)
/*
DESCRIPTION:
  Displays available years; Select years for stress
CALLED BY:
  stand_edit
NOTES:
  returns 0 to ISOPT if no years chosen
*/
{
   int i;
   menu_type menu;
   sed_type choose_sed;
   char yr_string[5];   /* e.g. "1990\n" */

   if ((menu = menu_Open()) == NULL)
   {
      printf ("unable to open menu");
      return(0);
   }
   win_width = 35;
   win_y = 4;
   win_x = 29;
   for (i = 0; i < NYEAR; i++ )
   {
      sprintf (yr_string, "%d", (ISTR_array[i].ISTR));
      menu_Printf(menu, "@f[# %s]\n", &ISTR_array[i].SELECTED, &check_funcs,
```

```
            yr_string );
    }
    menu_Flush(menu);
    if ((choose_sed = sed_Open(menu)) == NULL)
    {
       printf ("unable to open sed");
       return(0);
    }
    sed_SetLabel (choose_sed, 50);
    sed_SetWidth(choose_sed, win_width);
    sed_SetHeight(choose_sed, win_height);
    sed_SetPosition(choose_sed, win_y, win_x);
    sed_SetColors (choose_sed, CYAN_BLUE, WHITE_BLUE, YELLOW_BLACK);
    sed_SetShadow(choose_sed, 1);
    sed_SetShadowAttr(choose_sed, DGRAY_BLACK);
    sed_SetMouse(choose_sed, sedmou_GreedyTrack);
    sed_SetBorder(choose_sed, bd_mouse);
    sed_SetBorderTitle(choose_sed, "CHOOSE STRESS YEARS(use SPACE bar)");
    sed_Repaint(choose_sed);
    sed_Go(choose_sed);
    sed_Close(choose_sed);
    /* how many ISTRS are selected? */
    NSTRYR = 0;
    for (i = 0; i < NYEAR; i++ )
      if (ISTR_array[i].SELECTED == TRUE)
         NSTRYR = NSTRYR + 1;
    if (NSTRYR == 0)
    {
       beep ();
       prompt ("WARNING: No Stress Years chosen;  Stress option (ISOPT) will be reset to zero");
       return(0);
    }
    return(2);
} /* choose_ISTR_yrs ************************************************************/


void mem_check(boolean returned)
/*
DESCRIPTION:
  insures enough memory for menu_Printf
*/
{
    static int WARN_ONCE = 0;
    int key_stroke;

    if (returned == FALSE) {
       printf ("  ERROR: Unable to allocate memory!\n");
       if (WARN_ONCE == 0) {
          printf("  Continue with Program? (Y or N)\n");
          printf("  <Return> to continue (N is default)\n");
          key_stroke = getkey_stroke();
          switch (key_stroke)  {
             case 0xD:    /* carriage return */
             case 110: /* n */
             case 78: disp_Close();  /* N */
                      exit(0);
             default: ;
               break;
          }
          WARN_ONCE++;
       }
    }
} /*************************************************************************/


/*********************************/
int getkey_stroke(void)
/*********************************
DESCRIPTION:
  waits for and returns the next keystroke input
*/
{
    int c ;

    c = getch() ;           /* get single ASCII character */
    if(c == 0)              /* is it a non-ASCII extended code? */
      c = 0x100 + getch() ; /*   yes - use 256 + scan code */
    return(c) ;
}
```

```
/*************** HOST SPECIES VARIABLES HEADER FILE ************/
/* CSCAPE window variables */
extern int win_width,       /* window width */
    win_x,          /* window position (x coord) of upper left hand corner */
    win_y,          /* window position (y coord) of upper left hand corner */
    win_height;     /* window height */


/* GENERAL data file variables */
extern int NHOSTS;
extern char GMSUBM[2];

/* STAND data file variables */
extern float DLENBASE;

/* FILE ACCESS variables */
extern char speciesfile_name[13];
extern FILE *speciesfile;

extern MGMT_YRS *start_mgmt_yr_ptr, mgmt_yr_struct;
extern DEFOL_YRS *start_defol_yr_ptr;

/* USER INTERFACE variables */
extern int host_to_delete,
   tree_index;
extern boolean EDITS_MADE,
   CHOICE;
extern int units;


/* USER INTERFACE HOST SPECIES variables */
extern char *tree_strings[];
extern char tree_2chr[POSIBL_SPECIES][3];

extern TREE tree[MAX_HOSTS];  /* array of tree structures */

char tree_pcx[POSIBL_SPECIES][7] = {
  "WO.PCX","SO.PCX","CO.PCX","RO.PCX","BO.PCX",
  "RM.PCX","SM.PCX","ST.PCX","YB.PCX","PB.PCX",
  "SB.PCX","AB.PCX","BW.PCX","YP.PCX","FD.PCX",
  "AS.PCX","BC.PCX","WA.PCX","HI.PCX","WP.PCX",
  "BG.PCX","UD.PCX" };

int pcx_shown = SUCCESS;
int INIT_PCX_ONCE = 0;
extern int graphics_capable;
int pcx_being_displayed = FALSE;
```

```
/********************** HOST SPECIES EDIT, DELETE, ADD ********************/
/* Microsoft C 6.0 includes */
#include <stdio.h>
#include <string.h>
/* CSCAPE 3.2 includes */
#include <cscape.h>
#include <popdecl.h>
#include <pmwinobj.h>
#include <pcscan.h>
/* define file */
#include "defines.h"
/* variables and functions for host species routines */
#include "struct.h"
#include "host.h"
#include "options.h"
#define SMALL_PCX                0


/* FUNCTION PROTOTYPES - using CSCAPE */
int choose_host_edit(VOID *sdata, int idata);
int delete_species (VOID *sdata, int idata);
PRIVATE int host_edit(int index, int tree_index);
int add_species (VOID *sdata, int idata);
void verify_choice ( char msg1[50], char msg2[50] );
int field_grid_with_enter(sed_type sed,int scancode);


/* no CSCAPE */
PRIVATE void delete_host_values ( int host_to_delete );
PRIVATE void delete_last_linked_values();
int get_tree_index ( char tname[3] );
PRIVATE void open_species_file ( char file_name[13], int index, int tree_index );
void read_one_species (FILE *fptr, int index );
void mem_check(boolean returned);


#ifdef LEAF_PCX
/* EXTERNAL FUNCTIONS */
int attach_bob_to_pcx (int display_type, char pcx_name[13], int time_delay,
              byte win_color, byte bord_color, class_fptr win_border,
              bob_type *bob,
              int x1, int y1, int x2, int y2);
void close_pcx (void);
int initialize_pcx (void);
boolean display_pcx2(char *fname);
#endif


int choose_host_edit (VOID *sdata, int idata)
/*
DESCRIPTION:
  Displays available host species to edit. Select with space bar.
CALLED BY:
  framer menu EDIT TREE SPECIES
CALLS:
  get_tree_index, host_edit
*/
{
   int i;
   menu_type menu;
   sed_type choose_sed;
   boolean SELECTED[MAX_HOSTS];

   if ((menu = menu_Open()) == NULL)
   {
      printf ("unable to open menu");
      return(0);
   }
   win_width = 45;
   win_y = 8;
   win_x = 5;
   for (i = 0; i < NHOSTS; i++)  {
     SELECTED[i] = FALSE;
     menu_Printf(menu, " @f[# %s %s]\n", &SELECTED[i], &check_funcs,
       tree[i].TNAME, tree_strings[get_tree_index(tree[i].TNAME)] );
   }
   menu_Flush(menu);
   if ((choose_sed = sed_Open(menu)) == NULL) {
      printf ("unable to open sed");
      return(0);
   }
   sed_SetLabel (choose_sed, 41);
```

```
   sed_SetWidth(choose_sed, win_width);
   sed_SetPosition(choose_sed, win_y, win_x);
   sed_SetColors (choose_sed, CYAN_BLUE, WHITE_BLUE, YELLOW_BLACK);
   sed_SetShadowAttr(choose_sed, DGRAY_BLACK);
   sed_SetShadow(choose_sed, 1);
   sed_SetMouse(choose_sed, sedmou_GreedyTrack);
   sed_SetBorder(choose_sed, bd_title);
   sed_SetBorderTitle(choose_sed, " CHOOSE HOSTS TO EDIT (mark with SPACE bar)");
   sed_Repaint(choose_sed);
   sed_Go(choose_sed);
   for (i = 0; i < NHOSTS; i++)  {
     if (SELECTED[i] != FALSE)
        (void)host_edit( i, get_tree_index(tree[i].TNAME) );
   }
   sed_Close(choose_sed);
   return (0);
} /* choose_host_edit ********************************************************/


int delete_species (VOID *sdata, int idata)
/*
DESCRIPTION:
  displays available host species. Deletes number chosen.
CALLED BY:
  framer menu DELETE TREE SPECIES
CALLS:
  delete_host_values, get_tree_index
NOTES:
  CHOICE is global boolean
*/
{
   int i;
   menu_type menu;
   sed_type del_sp_sed;
   char range_string[6];
   char msg_string[40];

   host_to_delete = 0;
   sprintf ( range_string, "(0,%d)", NHOSTS);
   if ((menu = menu_Open()) == NULL)  {
      printf ("unable to open menu");
      return(0);
   }
   win_width = 30;
   win_height = NHOSTS+1;
   win_y = 7;
   win_x = 5;
   for (i = 0; i < NHOSTS; i++)
     menu_Printf(menu, "%d: %s %s%s", i+1,
       tree[i].TNAME, tree_strings[get_tree_index (tree[i].TNAME)], "\n");
   menu_Printf(menu, "Delete Host Number:@fd2[###]",
     &host_to_delete,&int_funcs,"Number of host to delete", range_string);
   menu_Flush(menu);
   if ((del_sp_sed = sed_Open(menu)) == NULL)  {
      printf ("unable to open sed");
      return(0);
   }
   sed_SetLabel (del_sp_sed, 43);
   sed_SetWidth(del_sp_sed, win_width);
   sed_SetPosition(del_sp_sed, win_y, win_x);
   sed_SetHeight(del_sp_sed, win_height);
   sed_SetColors (del_sp_sed, WHITE_RED, WHITE_BLUE, RED_WHITE);
   sed_SetShadow(del_sp_sed, 1);
   sed_SetShadowAttr(del_sp_sed, DGRAY_BLACK);
   sed_SetMouse(del_sp_sed, sedmou_GreedyClick);
   sed_SetBorder(del_sp_sed, bd_std);
   sed_SetBorderTitle(del_sp_sed, "    CHOOSE HOST TO DELETE  ");
   sed_Repaint(del_sp_sed);
   sed_Go(del_sp_sed);
   if (host_to_delete > 0) {
      if (NHOSTS == 1) {
         beep();
         pop_Prompt("CANNOT delete the last host! (add a host first)", -1, -1, -1, 45, WHITE_RED, bd_2);
      }
      else  {
         CHOICE = FALSE;
         sprintf(msg_string, "Delete %s?",
            tree_strings[get_tree_index(tree[host_to_delete-1].TNAME)]);
         verify_choice(msg_string, "WARNING: Deleted host's data cannot be retrieved");
         if (CHOICE == TRUE)  {
             EDITS_MADE = TRUE;
             if (host_to_delete < NHOSTS)
```

```
                delete_host_values ( host_to_delete );
             if (NHOSTS == MAX_HOSTS)
                /* get rid of linked list values at end of linked list arrays */
                delete_last_linked_values ();
             NHOSTS = NHOSTS - 1;    /* if host_to_delete == NHOSTS, simply decrement NHOSTS */
          }
       }
    }
    sed_Close(del_sp_sed);
    return (0);
} /* delete_species *******************************************************/


int get_tree_index ( char tname[3] )
/*
DESCRIPTION:
  returns the index of the tree from the list of 20 species
ARGUMENTS:
  tname is the 2 character species code
RETURNS:
  tree_index
SAMPLE CALL:
  get_tree_index ("RO")
CALLED BY:
  choose_host_edit, delete_species, mgmt_edit_for_one_yr,
  defol_edit_for_one_yr
*/
{
  int i,
      index;

  index = 0;
  for ( i=0; i < POSIBL_SPECIES; i++ )
    if (tname[0] == tree_2chr[i][0] &&
        tname[1] == tree_2chr[i][1] )
       index = i;
  return (index);
} /* get_tree_index *******************************************************/


int add_species (VOID *sdata, int idata)
/*
DESCRIPTION:
  adds a species; MAX_HOSTS species is maximum allowed
CALLED BY:
  framer menu ADD TREE SPECIES
CALLS:
  open_species_file, host_edit
*/
{
   int i, j;
   menu_type menu;
   sed_type add_sed;
   char BLANKCHR[2];
   char temp1_string [22];
   char temp2_string [22];
   char menu_tree_string [27]; /* "@f[" + tree_strings[i] + BLANKCHRs + "]\n" */

   if (NHOSTS < MAX_HOSTS)  {
     if ((menu = menu_Open()) == NULL)  {
        printf ("unable to open menu");
        return(0);
     }
     win_width = 45;
     win_y = 7;
     win_x = 2;
     win_height = 15;
     strcpy ( BLANKCHR, " ");
     for (i = 0; i < POSIBL_SPECIES; i++)  {
        strcpy (temp1_string, tree_strings[i]);
        strcpy (temp2_string, tree_strings[i]);
        /* put BLANKCHRs on end of string */
        for (j = 0; j < (17 - strlen(temp1_string)); j++)
           strcat (temp2_string, BLANKCHR);
        sprintf (menu_tree_string, " @f[%s]\n", temp2_string );
        menu_Printf(menu, menu_tree_string, NULL, &menu_funcs);
     }
     menu_Flush(menu);
     if ((add_sed = sed_Open(menu)) == NULL)  {
        printf ("unable to open sed");
        return(0);
     }
```

```
      sed_SetLabel (add_sed, 44);
      sed_SetWidth(add_sed, win_width);
      sed_SetHeight(add_sed, win_height);
      sed_SetPosition(add_sed, win_y, win_x);
      sed_SetColors (add_sed, CYAN_BLUE, WHITE_BLUE, YELLOW_CYAN);
      sed_SetShadow(add_sed, 1);
      sed_SetShadowAttr(add_sed, DGRAY_BLACK);
      sed_SetMouse(add_sed, sedmou_GreedyTrack);
      sed_SetBorder(add_sed, bd_mouse);
      sed_SetBorderTitle(add_sed, "Highlight with ARROW keys; Select with ENTER");
      sed_Repaint(add_sed);
      sed_Go(add_sed);
      if (sed_GetBaton(add_sed) > 0)  {
         /* edit new host-increment NHOSTS */
         tree_index = sed_GetBaton(add_sed)-1;
         strcpy (tree[NHOSTS].TNAME, tree_2chr[tree_index] );
         yellow_message("Reading host species from disk");
         open_species_file ( speciesfile_name, NHOSTS, tree_index );
         yellow_message(NULL);
         (void)host_edit(NHOSTS, tree_index);
         NHOSTS = NHOSTS+1;
      }
      sed_Close(add_sed);
   }
   else {
      beep();
      prompt(" Cannot add to list! (12 maximum)\n Delete one first");
   }
   return (0);
} /* add_species ********************************************************/


int host_edit (int hindex, int tree_index)
/*
DESCRIPTION:
  edits one host species
ARGUMENTS:
  hindex is the host index (1-MAX_HOSTS);
CALLED BY:
  choose_host_edit, add_species
*/
{
   menu_type menu;
   sed_type sed;
   float class_midpt[20];  /* diamater class midpoints */
   boolean COLD_greater_HOT = TRUE;
   int i, j, k;
   int ret;
   char species_string[30];
   bob_type bob;
#ifdef LEAF_PCX
   int pcx_width = 19;
   int pcx_height = 10;

  if (graphics_capable)
      if (0 == INIT_PCX_ONCE)  {
         pcx_shown = initialize_pcx();
         INIT_PCX_ONCE++;
      }
#endif

   if (units == SI)  {
      DLENBASE *= IN_to_CM;
      for (i = 0; i < 20; i++)
         tree[hindex].STEMS[i] *= AC_to_HA;
      tree[hindex].RECRUT *= AC_to_HA;
      tree[hindex].COLD *= FDD_to_CDD;
      tree[hindex].HOT  *= FDD_to_CDD;
      tree[hindex].SLOWD *= IN_to_CM;
      tree[hindex].DMAX *= IN_to_CM;
      tree[hindex].HMAX *= IN_to_CM;
      tree[hindex].SURFAR *= SQIN_OZ_SQCM_GM;
   }
   EDITS_MADE = TRUE;
   win_y = 1;
   win_x = 5;
   win_height = disp_GetHeight()-3;
   win_width = 68;

#ifdef LEAF_PCX
   if (pcx_shown == SUCCESS && graphics_capable)
      pcx_shown = attach_bob_to_pcx(SMALL_PCX, tree_pcx[tree_index], 0,
```

```
                    0XB2, 0xB2, bd_1,
                    &bob,
                    /* x1   , y1,   x2, y2 */
                    0, 0, pcx_width, pcx_height);
#endif

   if ((menu = menu_Open()) == NULL)  {
       printf ("unable to open menu");
      return(0);
   }

#ifdef LEAF_PCX
   /* show pcx bob */
   if (pcx_shown == SUCCESS && graphics_capable)
      mem_check(menu_Printf(menu, "@[22, ]@fpb[]@[15,\n]", NULL, &bob_funcs, bob));
#endif
   sprintf (species_string, "[%s]", tree_strings[tree_index]);
   mem_check(menu_Printf(menu,"   Species: @fd2%s", NULL,
     &menu_funcs, species_string, "Help with species (Press Enter)", "1"));
   mem_check(menu_Printf(menu, "          Code: %s",
     tree_2chr[tree_index] ));
   mem_check(menu_Printf(menu, "      Host Code: @fp[###]\n",
     &tree[hindex].host_code, &int_funcs));
   mem_check(menu_Printf(menu,"@[76,%c]\n", 205)); /* double line */
   mem_check(menu_Printf(menu,"@[17, ]@a[0x1F]INITIAL CONDITIONS@a[0x17]\n"));
   mem_check(menu_Printf(menu,"@[20, ]@a[0x1B]Stem Counts@a[0x17]\n"));
   if (units == SI)
      mem_check(menu_Printf(menu, "Indexed by diameter class midpoint;  class width(cm.) = %2.1f\n", DLENBASE));
   else
      mem_check(menu_Printf(menu, "Indexed by diameter class midpoint;  class width(in.) = %2.1f\n", DLENBASE));
   for (i = 0; i < 20; i++)  {
     class_midpt[i] = i * DLENBASE + (DLENBASE/2.0);
     mem_check(menu_Printf(menu, "%5.1f:@fd2[@[8,#]]", class_midpt[i],
       &tree[hindex].STEMS[i], &sfloat_funcs,"Number of stems in diameter class; > or = 0; STEMS", "(0,)"));
     if (((i+1) % 4) == 0)  /* skip line */
        mem_check(menu_Printf(menu, "\n"));
   }
   mem_check(menu_Printf(menu,"@[76,%c]\n", 196)); /*line*/
   mem_check(menu_Printf(menu,"@[14, ]@a[0x1B]Defoliation History Index@a[0x17]\n"));
   mem_check(menu_Printf(menu,"0:None  1: 1-30 per cent    2: 31-65 per cent    3: >65 per cent\n"));
   mem_check(menu_Printf(menu,"@[23, ]Overstory      Understory\n" ));
   for ( k=0; k < 2; k++ )  {
     for ( j=0; j < 2; j++ )  {
        if ((0 == k) && (0 == j))
           mem_check(menu_Printf(menu, "Sample year  "));
        if ((1 == k) && (0 == j))
           mem_check(menu_Printf(menu, "Previous year"));
        mem_check(menu_Printf(menu, "            @fd2[###]",
          &tree[hindex].IDHIST[j][k],&int_funcs,"0=None    1=Light    2=Moderate    3=Heavy;  IDHIST","(0,3)"));
     }
     mem_check(menu_Printf(menu, "\n"));
   }
   mem_check(menu_Printf(menu,"@[76,%c]\n", 205));
   mem_check(menu_Printf(menu, "\n"));
   mem_check(menu_Printf(menu,"                  @a[0x1F]PARAMETER VALUES@a[0x17]\n"));
   mem_check(menu_Printf(menu, "Maximum Age: @fd2[######]\n",
     &tree[hindex].AGEMAX,&sfloat_funcs,"Used to calculate base tree mortality rate; >0; AGEMAX", "(0.1,600)"));
   mem_check(menu_Printf(menu, "Maximum Height: @fd2[########]\n",
     &tree[hindex].HMAX,&sfloat_funcs,"Maximum height for eqtn 15; > 0; HMAX", "(0,)"));
   mem_check(menu_Printf(menu, "Maximum Diameter: @fd2[########]\n",
     &tree[hindex].DMAX,&sfloat_funcs,"Maximum diameter for eqtn 15; > 0; DMAX", "(0,)"));
   mem_check(menu_Printf(menu, "Diameter Growth: @fd2[#####]\n",
     &tree[hindex].GROWR,&sfloat_funcs,"Base Growth Rate; GROWR", "(0,)"));
   mem_check(menu_Printf(menu, "Slow Growth Threshold: @fd2[#########]\n",
     &tree[hindex].SLOWD,&sfloat_funcs,"Diameter growth below which additional mortality induced; >=0;SLOWD",
"(0,)"));
   mem_check(menu_Printf(menu, "Minimum Degree-Days: @fd2[########]\n",
     &tree[hindex].COLD,&sfloat_funcs,"Accumulated Minimum Day Degrees expected for this species; COLD",
"(0,)"));
   mem_check(menu_Printf(menu, "Maximum Degree-Days: @fd2[########]\n",
     &tree[hindex].HOT,&sfloat_funcs,"Accumulated Maximum Day Degrees expected for this species; HOT", "(0,)"));
   mem_check(menu_Printf(menu, "Relative Stocking Class: @fd2[###]\n",
     &tree[hindex].ISTOKG,&int_funcs,"Table 2; > 0; ISTOKG", "(0,)"));
   mem_check(menu_Printf(menu,"@a[0x1B]Shade Tolerance Indexes@a[0x17]\n"));
   mem_check(menu_Printf(menu, "    Recruitment: @fd2[###]\n",
     &tree[hindex].ISHADE[0],&int_funcs,"Recruitment shade tolerance index (1-6); ISHADE[][1]", "(1,6)"));
   mem_check(menu_Printf(menu, "    Diameter growth: @fd2[###]\n\n",
     &tree[hindex].ISHADE[1],&int_funcs,"Dia. growth shade tolerance index (1-6); ISHADE[][2]", "(1,6)"));
   mem_check(menu_Printf(menu, "Recruitment: @fd2[########]\n",
     &tree[hindex].RECRUT,&sfloat_funcs,"Max number stems recruited; > 0 (RECRUT)", "(0,)"));
   mem_check(menu_Printf(menu, "Biomass/Surface Area: @fd2[##########]\n",
     &tree[hindex].SURFAR,&sfloat_funcs,"Conversion factor; > 0; SURFAR", "(0,)"));
```

```
   mem_check(menu_Printf(menu,"\n"));
   mem_check(menu_Printf(menu,"@a[0x1B]Tree Mortality Following Heavy Defoliation: >65 per cent@a[0x17]\n" ));
   mem_check(menu_Printf(menu,"SITE          1YR        2YR        3YRs or more\n" ));
   for ( k=0; k < 3; k++ )  {
     for ( j=0; j < 3; j++ )  {
        if(j == 0)  {
           if(k == 0)
              mem_check(menu_Printf(menu,"Dry      " ));
           else if (k == 1)
              mem_check(menu_Printf(menu,"Moderate " ));
           else /* k == 2 */
              mem_check(menu_Printf(menu,"Moist    " ));
        }
        mem_check(menu_Printf(menu, "      @fd2[#####]",
          &tree[hindex].FDIE[j][k],&sfloat_funcs,"Annual proportion dying after defoliation; 0->1; FDIE",
"(0,1)"));
     }
     mem_check(menu_Printf(menu, "\n"));
   }
   mem_check(menu_Printf(menu,"\n"));
   /* edit variables if gypsy moth submodel being run */
   if ((strcmp("T", GMSUBM)==0) || (strcmp("t", GMSUBM)==0))  {
      mem_check(menu_Printf(menu, "      @a[0x1B]Foliage Biomass Predictor Coefficients@a[0x17]\n" ));
      mem_check(menu_Printf(menu, "Foliage parameter F1: @fd2[#######]\n",
         &tree[hindex].F1,&sfloat_funcs,"See table 2; > 0; F1", "(0,)"));
      mem_check(menu_Printf(menu, "Foliage parameter F2: @fd2[#######]\n",
         &tree[hindex].F2,&sfloat_funcs,"See table 2; > 0; F2", "(0,)"));
      mem_check(menu_Printf(menu,"\n"));
      mem_check(menu_Printf(menu, "@[14, ]Resting Sites\n" ));
      mem_check(menu_Printf(menu, "On boles of overstory trees: @fd2[####]\n",
        &tree[hindex].RESTIN[0],&sfloat_funcs,"Max. larvae/100 sq. cm. on lower boles; > 0; RESTIN[][1]",
"(0,)"));
      mem_check(menu_Printf(menu, "On understory trees & canopy of overstory trees: @fd2[####]\n",
        &tree[hindex].RESTIN[1],&sfloat_funcs,"Max. larvae/100 sq. cm. on upper boles; > 0; RESTIN[][2]",
"(0,)"));
      mem_check(menu_Printf(menu,"\n"));
      mem_check(menu_Printf(menu, "      Live Crown Ratio predictor coefficients\n" ));
      mem_check(menu_Printf(menu, "CR1: @fd2[#####]\n",
        &tree[hindex].CR1,&sfloat_funcs,"Coeficient CR1; > 0", "(0,)"));
      mem_check(menu_Printf(menu, "CR2: @fd2[#####]\n",
        &tree[hindex].CR2,&sfloat_funcs,"Coeficient CR2; > 0", "(0,)"));
      mem_check(menu_Printf(menu, "CR3: @fd2[#####]\n",
        &tree[hindex].CR3,&sfloat_funcs,"Coeficient CR3; > 0", "(0,)"));
      mem_check(menu_Printf(menu, "CR4: @fd2[#####]\n",
        &tree[hindex].CR4,&sfloat_funcs,"Coeficient CR4; > 0", "(0,)"));
   }
   menu_Flush(menu);
   if ((sed = sed_Open(menu)) == NULL)  {
      printf ("unable to open sed");
      return(0);
   }
   sed_SetLabel(sed, 45);
   sed_SetWidth(sed, win_width);
   sed_SetHeight(sed, win_height);
   sed_SetPosition(sed, win_y, win_x);
   sed_SetColors (sed, CYAN_BLUE, WHITE_BLUE, YELLOW_CYAN);
   sed_SetShadowAttr(sed, DGRAY_BLACK);
   sed_SetMouse(sed, sedmou_GreedyTrack);
   sed_SetBorder(sed, bd_mouse);
   sed_SetBorderTitle(sed, tree_strings[tree_index]);
   sed_SetSpecial(sed, field_grid_with_enter);

#ifdef LEAF_PCX
   /* display pcx file */
   if (pcx_shown == SUCCESS && graphics_capable)  {
      if (!display_pcx2(tree_pcx[tree_index]))  {
         /* NO DISPLAY */
         pcx_being_displayed = FALSE;
         prompt ("Not enough memory to display graphic.");
      }
      else
         pcx_being_displayed = TRUE;
   }
#endif
   sed_Repaint(sed);
   ret = sed_Go(sed);
   do  {
      if (tree[hindex].COLD > tree[hindex].HOT)  {
         beep();
         prompt("                    ERROR\n Maximum Day Degrees must be greater\n than Minimum Day Degrees.");
         sed_GotoField(sed, 28);
         sed_Repaint(sed);
```

```
            sed_Go(sed);
         }
         else
            COLD_greater_HOT = FALSE;
         if (ret == 1) {
            help_Show(44, tree_index + 1);  /* help on each tree */
            sed_Repaint(sed);
            ret = sed_Go(sed);
         }
      }
      while (COLD_greater_HOT || ret == 1);
#ifdef LEAF_PCX
      if (pcx_shown == SUCCESS  &&  graphics_capable)
         close_pcx();
#endif
      sed_Close(sed);
      if (units == SI)  {
         DLENBASE /= IN_to_CM;
         for (i = 0; i < 20; i++)
            tree[hindex].STEMS[i] /= AC_to_HA;
         tree[hindex].RECRUT /= AC_to_HA;
         tree[hindex].COLD /= FDD_to_CDD;
         tree[hindex].HOT  /= FDD_to_CDD;
         tree[hindex].SLOWD /= IN_to_CM;
         tree[hindex].DMAX /= IN_to_CM;
         tree[hindex].HMAX /= IN_to_CM;
         tree[hindex].SURFAR /= SQIN_OZ_SQCM_GM;
      }
      return (0);
} /* host_edit ************************************************************/


/*****************************************/
 int    field_grid_with_enter(sed_type sed,int scancode)
/*****************************************
DESCRIPTION: Moves cursor across fields.
*/
{
    switch (scancode){
       case LEFT: sed_LeftField(sed); return(TRUE);
       case RIGHT: sed_RightField(sed); return(TRUE);
       case UP: sed_UpField(sed); return(TRUE);
       case DOWN: sed_DownField(sed); return(TRUE);
       default: return (FALSE);
    }
}


void delete_host_values ( int host_to_delete )
/*
DESCRIPTION:
  replaces deleted host position; reorder species arrays
  replaces deleted host positions in MGMT_YRS linked list; reorder cutmin, cutmax, action
ARGUMENTS:
  host_to_delete is the specified host species to remove
SAMPLE CALL:
  delete_host_values ( host_to_delete );
CALLED BY:
  delete_species
NOTES:
  host_to_delete is global, passed to emphasize importance in fnc
*/
{
  int i, j, k;
  MGMT_YRS *search_mgmt_ptr;
  DEFOL_YRS *search_defol_ptr;

  yellow_message("Deleting Host");
  for (i = host_to_delete; i < NHOSTS; i++)  {
     strcpy (tree[i-1].TNAME, tree[i].TNAME);
     for (j = 0; j < 20; j++)
        tree[i-1].STEMS[j] = tree[i].STEMS[j];
     tree[i-1].ISHADE[0] = tree[i].ISHADE[0];
     tree[i-1].ISHADE[1] = tree[i].ISHADE[1];
     tree[i-1].RESTIN[0] = tree[i].RESTIN[0];
     tree[i-1].RESTIN[1] = tree[i].RESTIN[1];
     tree[i-1].GROWR = tree[i].GROWR;
     tree[i-1].COLD  = tree[i].COLD;
     tree[i-1].HOT   = tree[i].HOT;
     tree[i-1].AGEMAX= tree[i].AGEMAX;
     tree[i-1].SLOWD = tree[i].SLOWD;
     tree[i-1].CR1   = tree[i].CR1;
```

```
      tree[i-1].CR2   = tree[i].CR2;
      tree[i-1].CR3   = tree[i].CR3;
      tree[i-1].CR4   = tree[i].CR4;
      tree[i-1].DMAX  = tree[i].DMAX;
      tree[i-1].HMAX  = tree[i].HMAX;
      tree[i-1].F1    = tree[i].F1;
      tree[i-1].F2    = tree[i].F2;
      tree[i-1].SURFAR = tree[i].SURFAR;
      tree[i-1].ISTOKG = tree[i].ISTOKG;
      tree[i-1].RECRUT = tree[i].RECRUT;
      for ( k=0; k < 3; k++ )
        for ( j=0; j < 3; j++ )
          tree[i-1].FDIE[j][k] = tree[i].FDIE[j][k];
      for ( k=0; k < 2; k++ )
        for ( j=0; j < 2; j++ )
          tree[i-1].IDHIST[j][k] = tree[i].IDHIST[j][k];
  }

  /* replace deleted host positions in MGMT_YRS linked list */
  search_mgmt_ptr = start_mgmt_yr_ptr;
  while (search_mgmt_ptr != NULL_MGMT_PTR)  {
     for (i = host_to_delete; i < MAX_HOSTS; i++)  {
        search_mgmt_ptr->cutmin[i-1] = search_mgmt_ptr->cutmin[i];
        search_mgmt_ptr->cutmax[i-1] = search_mgmt_ptr->cutmax[i];
        search_mgmt_ptr->action[i-1] = search_mgmt_ptr->action[i];
     }
     search_mgmt_ptr = search_mgmt_ptr->next;
  }

  /* replace deleted host positions in DEFOL_YRS linked list */
  search_defol_ptr = start_defol_yr_ptr;
  while (search_defol_ptr != NULL_DEFOL_PTR)  {
     for (i = host_to_delete; i < MAX_HOSTS; i++)  {
        search_defol_ptr->defol_amt[0][i-1] = search_defol_ptr->defol_amt[0][i];
        search_defol_ptr->defol_amt[1][i-1] = search_defol_ptr->defol_amt[1][i];
     }
     search_defol_ptr = search_defol_ptr->next;
  }

  yellow_message(NULL);
} /*  delete_host_values ******************************************************/


void delete_last_linked_values()
/*
DESCRIPTION:
  eliminate last array elements from linked list when NHOST = MAX_HOSTS
CALLED BY:
  delete_species
*/
{
  MGMT_YRS *mgmt_ptr;
  DEFOL_YRS *defol_ptr;

  /* erase last host positions in MGMT_YRS linked list */
  mgmt_ptr = start_mgmt_yr_ptr;
  while (mgmt_ptr != NULL_MGMT_PTR)  {
     mgmt_ptr->cutmin[MAX_HOSTS-1] = 0;
     mgmt_ptr->cutmax[MAX_HOSTS-1] = 0;
     mgmt_ptr->action[MAX_HOSTS-1] = 0;
     mgmt_ptr = mgmt_ptr->next;
  }

  /* erase last host positions in DEFOL_YRS linked list */
  defol_ptr = start_defol_yr_ptr;
  while (defol_ptr != NULL_DEFOL_PTR)  {
     defol_ptr->defol_amt[0][MAX_HOSTS-1] = 0;
     defol_ptr->defol_amt[1][MAX_HOSTS-1] = 0;
     defol_ptr = defol_ptr->next;
  }
} /* delete_last_linked_values*********************************************/


void open_species_file ( char file_name[13], int index, int tree_index )
/*
DESCRIPTION:
  open species file and call read_one_species
ARGUMENTS:
  file_name is disk file with 20 default tree species parameters
  index is host index: can be from 0 to 5 (MAX_HOSTS possible)
  tree_index can range from 0 to 19 (20 possible)
SAMPLE CALL:
```

```
  open_species_file ( speciesfile_name, i, get_tree_index( TNAME[i] ));
CALLED BY:
  add_species
CALLS:
  read_one_species
NOTES:
  STEMS are given values of 0 except for STEMS[0] = 1
*/
{
  char buffer[100];
  int i, j;

  speciesfile = fopen ( file_name, "r" );
  if ( speciesfile == ( FILE * ) NULL )
      printf ("%s file could not be opened.\n", file_name );

  /* skip to appropriate place in trees file */
  for ( i=0; i < (ROWS_FOR_ONE_TREE * tree_index); i++ )
    fscanf (speciesfile, "%[^\n]\n", buffer);

  read_one_species ( speciesfile, index );
  fclose(speciesfile);

  /* assign STEMS rather than read from file */
  tree[index].STEMS[0] = 1;
  for ( j=1; j < 20; j++ )
    tree[index].STEMS[j] = 0;
} /* open_species_file ************************************************/
```

```
/***************** DEFOLIATION VARIABLES HEADER FILE *************/
/* CSCAPE window variables */
extern int win_width, /* window width */
    win_x,            /* window position (x coord) of upper left hand corner */
    win_y,            /* window position (y coord) of upper left hand corner */
    win_height;       /* window height */


/* GENERAL data file variables */
extern int NYEAR,
    NHOSTS,
    ISYEAR;


/* individual HOST SPECIES data file variables */
extern TREE tree[MAX_HOSTS];  /* array of tree structures */
extern char LDEFOL[2];


/* DEFOLIATION data file variables used with linked list */
extern DEFOL_YRS *start_defol_yr_ptr, defol_yr_struct;

extern int LOCATED,
   status,          /* == SUCCESSFUL_DELETION, KEY_NOT_FOUND, LIST_EMPTY, etc. */
   number_of_defol_yrs;     /* number of defoliation years */


/* USER INTERFACE variables */
extern boolean EDITS_MADE,
   CHOICE;


/* USER INTERFACE HOST SPECIES variables */
extern char *tree_strings[];
```

```
/********************** DEFOLIATION ROUTINES *************************/
/* Microsoft C 6.0 includes */
#include <stdio.h>
#include <string.h>
#include <malloc.h>  /* free fnc */
/* CSCAPE 3.2 includes */
#include <cscape.h>
#include <popdecl.h>
/* define file */
#include "defines.h"
/* variables and functions for Defoliation routines */
#include "struct.h"
#include "defol.h"

/* FUNCTION PROTOTYPES */
/* routines using CSCAPE function calls */
void verify_choice ( char msg1[50], char msg2[50] );
int choose_defol_yr_edit (VOID *sdata, int idata);
int add_defol_yr (VOID *sdata, int idata);
int delete_defol_yr (VOID *sdata, int idata);
PRIVATE void defol_edit_for_one_yr ( DEFOL_YRS *struct_ptr );
int remove_all_defol (VOID *sdata, int idata);

/* no CSCAPE */
PRIVATE void initialize_defol_yr_struct ( DEFOL_YRS *struct_ptr );
DEFOL_YRS *linked_defol_yr_insert (DEFOL_YRS *list_start, DEFOL_YRS new_str );
void linked_defol_yr_extract (DEFOL_YRS *list_start, key_type search_key,
                              DEFOL_YRS *extracted_str, int *LOCATED );
DEFOL_YRS *linked_defol_yr_delete (DEFOL_YRS *list_start, key_type old_key,
                                   int *status );
void linked_defol_yr_duplication (DEFOL_YRS *list_start, int key, int *status );
int get_tree_index ( char tname[3] );


int add_defol_yr (VOID *sdata, int idata)
/*
DESCRIPTION:
  adds a defoliation year
CALLED BY:
  framer menu ADD DEFOLIATION YEAR
CALLS:
  initialize_defol_yr_structure, linked_defol_yr_duplication, defol_edit_for_one_yr,
  linked_defol_yr_insert
NOTES:
  MAX_DEFOL_YRS is maximum allowed
*/
{
   menu_type menu;
   sed_type add_yr_sed;
   char range_string[25];   /* e.g. "(1978,2000)\n"  */
   char start_yr_string[5];      /*  "1990\n" */
   char end_yr_string[5];
   int ending_yr;

   if (number_of_defol_yrs < MAX_DEFOL_YRS )
   {
      strcpy ( LDEFOL, "T");
      defol_yr_struct.defol_yr_key = 0;
      ending_yr = ISYEAR + NYEAR - 1;
      sprintf (range_string, "(0,0)(%d,%d)", ISYEAR, ending_yr);
      sprintf (start_yr_string, "%d", ISYEAR);
      sprintf (end_yr_string, "%d", ending_yr);
      if ((menu = menu_Open()) == NULL)
      {
         printf ("unable to open menu");
         return(0);
      }
      win_width = 66;
      win_y = 7;
      win_x = 4;
      menu_Printf(menu, "Starting Year: %s ",
         start_yr_string);
      menu_Printf(menu, "  Ending Year: %s \n",
         end_yr_string);
      menu_Printf(menu, "Defoliation Year: @fd2[#####]\n",
         &defol_yr_struct.defol_yr_key, &int_funcs,"Year for defoliation;", range_string);
      menu_Flush(menu);
      if ((add_yr_sed = sed_Open(menu)) == NULL)
      {
         printf ("unable to open sed");
         return(0);
      }
```

```
      sed_SetLabel (add_yr_sed, 37);
      sed_SetWidth(add_yr_sed, win_width);
      sed_SetPosition(add_yr_sed, win_y, win_x);
      sed_SetColors (add_yr_sed, CYAN_BLUE, WHITE_BLUE, YELLOW_CYAN);
      sed_SetShadow(add_yr_sed, 1);
      sed_SetShadowAttr(add_yr_sed, DGRAY_BLACK);
      sed_SetMouse(add_yr_sed, sedmou_GreedyClick);
      sed_SetBorder(add_yr_sed, bd_std);
      sed_SetBorderTitle(add_yr_sed, "            ADD A DEFOLIATION YEAR");
      sed_Repaint(add_yr_sed);
      sed_Go(add_yr_sed);
      /* was year entered? */
      if (defol_yr_struct.defol_yr_key > 0)
      {
         /* does duplicate year exist? */
         linked_defol_yr_duplication (start_defol_yr_ptr, defol_yr_struct.defol_yr_key, &status );
         if (status == NO_DUPLICATE_KEY)
         {
            initialize_defol_yr_struct ( &defol_yr_struct );
            defol_edit_for_one_yr ( &defol_yr_struct );
            /* insert the edited structure into the linked_list */
            start_defol_yr_ptr = linked_defol_yr_insert (start_defol_yr_ptr, defol_yr_struct );
            number_of_defol_yrs = number_of_defol_yrs + 1;
         }
         else /* status == DUPLICATE_KEY */
         {
            beep();
            prompt (" Duplicate defoliation year entered");
         }
      }
      sed_Close(add_yr_sed);
   }
   else
   {
     beep();
     pop_Prompt(" No more defoliation years to enter! (100 maximum)", -1, -1, -1, 35, WHITE_RED, bd_2);
   }
   return (0);
} /* add_defol_yr ********************************************************/


void defol_edit_for_one_yr ( DEFOL_YRS *struct_ptr )
/*
DESCRIPTION:
   Stand defoliation parameter editing for one defoliation year
ARGUMENTS:
   struct_ptr is a pointer to the stand defoliation structure DEFOL_YRS
CALLED BY:
   add_defol_yr, choose_defol_yr_edit
CALLS:
   get_tree_index
*/
{
   int i;
   menu_type menu;
   sed_type defol_edit_sed;
   char yr_string[5];   /* e.g. "1990\n" */

   EDITS_MADE = TRUE;
   sprintf (yr_string, "%d", struct_ptr->defol_yr_key);
   if ((menu = menu_Open()) == NULL)
      printf ("unable to open menu");
   win_width = 46;
   win_y = 12;
   win_x = 26;
   menu_Printf(menu, " YEAR: @a[0x1F]%s@a[0x17]\n",
     yr_string);
   menu_Printf(menu, "    Overstory   Understory\n");
   for (i = 0; i < NHOSTS; i++)
   {
     menu_Printf(menu, " %s ",
       tree[i].TNAME);
     menu_Printf(menu,"   @fd2[####]",
       &struct_ptr->defol_amt[0][i], &sint_funcs,"Defoliation (%); 0->100;", "(0,100)");
     menu_Printf(menu,"     @fd2[####]",
       &struct_ptr->defol_amt[1][i], &sint_funcs,"Defoliation (%); 0->100;", "(0,100)");
     menu_Printf(menu, "    %s%s%s\n",
       "(", tree_strings[get_tree_index(tree[i].TNAME)], ")");
   }
   menu_Flush(menu);
   if ((defol_edit_sed = sed_Open(menu)) == NULL)
      printf ("unable to open sed");
```

```
   sed_SetLabel (defol_edit_sed, 38);
   sed_SetWidth(defol_edit_sed, win_width);
   sed_SetPosition(defol_edit_sed, win_y, win_x);
   sed_SetColors (defol_edit_sed, CYAN_BLUE, WHITE_BLUE, YELLOW_CYAN);
   sed_SetShadow(defol_edit_sed, 1);
   sed_SetShadowAttr(defol_edit_sed, DGRAY_BLACK);
   sed_SetMouse(defol_edit_sed, sedmou_GreedyTrack);
   sed_SetBorder(defol_edit_sed, bd_std);
   sed_SetBorderTitle(defol_edit_sed, "               DEFOLIATION");
   sed_SetSpecial(defol_edit_sed, inter_field_grid);
   sed_Repaint(defol_edit_sed);
   sed_Go(defol_edit_sed);
   sed_Close(defol_edit_sed);
} /* defol_edit_for_one_yr ************************************************/


int delete_defol_yr (VOID *sdata, int idata)
/*
DESCRIPTION:
  displays available defoliation years; Deletes year chosen
CALLED BY:
  framer menu DELETE DEFOLIATION YEAR
CALLS:
  verify_choice, linked_defol_yr_extract, linked_defol_yr_delete
*/
{
   menu_type menu;
   sed_type del_defol_sed;
   char yr_string[5];   /* e.g. "1990\n" */
   int yr_key_to_delete = 0;
   DEFOL_YRS *search_ptr;

   search_ptr = start_defol_yr_ptr;
   if (search_ptr != NULL_DEFOL_PTR)  {
      if ((menu = menu_Open()) == NULL)  {
         printf ("unable to open menu");
         return(0);
      }
      win_width = 36;
      win_y = 7;
      win_x = 5;
      if ((number_of_defol_yrs+1) < (disp_GetHeight()-10))
         win_height = number_of_defol_yrs+1;
      else
         win_height = disp_GetHeight()-10;
      while (search_ptr != NULL_DEFOL_PTR)  {
         sprintf (yr_string, "%d", search_ptr->defol_yr_key);
         menu_Printf(menu, " %s %s",
           yr_string, "\n");
         search_ptr = search_ptr->next;
      }
      menu_Printf(menu, "Delete Year:@fd[#####]",
         &yr_key_to_delete, &int_funcs,"Year to delete");
      menu_Flush(menu);
      if ((del_defol_sed = sed_Open(menu)) == NULL)  {
         printf ("unable to open sed");
         return(0);
      }
      sed_SetLabel (del_defol_sed, 39);
      sed_SetWidth(del_defol_sed, win_width);
      sed_SetHeight(del_defol_sed, win_height);
      sed_SetPosition(del_defol_sed, win_y, win_x);
      sed_SetColors (del_defol_sed, WHITE_RED, WHITE_BLUE, RED_WHITE);
      sed_SetShadow(del_defol_sed, 1);
      sed_SetShadowAttr(del_defol_sed, DGRAY_BLACK);
      sed_SetMouse(del_defol_sed, sedmou_GreedyClick);
      sed_SetBorder(del_defol_sed, bd_std);
      sed_SetBorderTitle(del_defol_sed, "CHOOSE DEFOLIATION YEAR TO DELETE");
      sed_Repaint(del_defol_sed);
      sed_Go(del_defol_sed);
      if (yr_key_to_delete > 0)  {
         /*initialized so struct can be sent to extract to check for existence*/
         defol_yr_struct.defol_yr_key = 1;
         initialize_defol_yr_struct ( &defol_yr_struct );
         linked_defol_yr_extract (start_defol_yr_ptr, yr_key_to_delete, &defol_yr_struct,
                                  &LOCATED);
         if (LOCATED == FALSE)  {
            beep();
            prompt ("defoliation year not found in list");
         }
```

```
         else  /* LOCATED */
         {
            CHOICE = FALSE;
            verify_choice ( "Delete indicated Defoliation Year?",
                            "WARNING: Defoliation Year will be lost");
            if (CHOICE == TRUE)  {
               EDITS_MADE = TRUE;
               start_defol_yr_ptr = linked_defol_yr_delete (start_defol_yr_ptr, yr_key_to_delete,
                                                            &status);
               if (status == KEY_NOT_FOUND)
                  prompt ("defoliation year not found in list");
               else  {
                  number_of_defol_yrs = number_of_defol_yrs - 1;
                  if (number_of_defol_yrs == 0)
                     strcpy ( LDEFOL, "F");
               }
            }
         }
      }
      sed_Close(del_defol_sed);
   }
   else
   {
      beep();
      prompt ("No Defoliation Years to Delete!");
   }
   return (0);
} /* delete_defol_yr ********************************************************/


int choose_defol_yr_edit (VOID *sdata, int idata)
/*
DESCRIPTION:
  Displays available defoliation years to edit; Select from list with space bar
CALLED BY:
  framer menu EDIT DEFOLIATION YEAR
CALLS:
  defol_edit_for_one_yr
*/
{
   int i, j;
   menu_type menu;
   sed_type choose_sed;
   boolean SELECTED[MAX_DEFOL_YRS];
   char yr_string[5];   /* e.g. "1990\n" */
   DEFOL_YRS *search_ptr;

   search_ptr = start_defol_yr_ptr;
   if (search_ptr != NULL_DEFOL_PTR)  {
      if ((menu = menu_Open()) == NULL)  {
         printf ("unable to open menu");
         return(0);
      }
      win_width = 47;
      win_y = 7;
      win_x = 5;
      if (number_of_defol_yrs < (disp_GetHeight()-10))
         win_height = number_of_defol_yrs;
      else
         win_height = disp_GetHeight()-10;
      for (j = 0; j < MAX_DEFOL_YRS; j++)
         SELECTED[j] = FALSE;
      i = 0;
      while (search_ptr != NULL_DEFOL_PTR)  {
         sprintf (yr_string, "%d", search_ptr->defol_yr_key);
         menu_Printf(menu, "@f[# %s]\n", &SELECTED[i], &check_funcs,
           yr_string );
         search_ptr = search_ptr->next;
         i = i + 1;
      }
      menu_Flush(menu);
      if ((choose_sed = sed_Open(menu)) == NULL)  {
         printf ("unable to open sed");
         return(0);
      }
      sed_SetLabel (choose_sed, 40);
      sed_SetWidth(choose_sed, win_width);
      sed_SetPosition(choose_sed, win_y, win_x);
      sed_SetHeight(choose_sed, win_height);
      sed_SetColors (choose_sed, CYAN_BLUE, WHITE_BLUE, YELLOW_BLACK);
      sed_SetShadow(choose_sed, 1);
      sed_SetShadowAttr(choose_sed, DGRAY_BLACK);
```

```
      sed_SetMouse(choose_sed, sedmou_GreedyTrack);
      sed_SetBorder(choose_sed, bd_title);
      sed_SetBorderTitle(choose_sed, "CHOOSE DEFOLIATION YEARS TO EDIT (use SPACE bar)");
      sed_Repaint(choose_sed);
      sed_Go(choose_sed);

      search_ptr = start_defol_yr_ptr;
      i = 0;
      while (search_ptr != NULL_DEFOL_PTR)  {
         if (SELECTED[i] != FALSE)
            defol_edit_for_one_yr ( search_ptr );
         search_ptr = search_ptr->next;
         i = i + 1;
      }
      sed_Close(choose_sed);
   }
   else  {
      beep();
      prompt ("Add a defoliation year first");
   }
   search_ptr = start_defol_yr_ptr;
   return (0);
} /* choose_defol_yr_edit ************************************************************/


int remove_all_defol (VOID *sdata, int idata)
/*
DESCRIPTION:
  deletes all defoliation years
CALLED BY:
  framer menu REMOVE ALL DEFOLIATION
CALLS:
  verify_choice
*/
{
   DEFOL_YRS *temp_ptr;

   temp_ptr = start_defol_yr_ptr;
   if (temp_ptr != NULL_DEFOL_PTR)  {
      CHOICE = FALSE;
      verify_choice ( "Remove all Defoliation Years?",
                      "WARNING: All Defoliation will be lost");
      if (CHOICE == TRUE)  {
         while (start_defol_yr_ptr != NULL_DEFOL_PTR)  {
            /* use temp_ptr to dispose of unwanted str */
            temp_ptr = start_defol_yr_ptr;
            start_defol_yr_ptr = start_defol_yr_ptr->next;
            free (temp_ptr);
         }
         start_defol_yr_ptr = NULL_DEFOL_PTR;
         number_of_defol_yrs = 0;
         strcpy ( LDEFOL, "F");

         status = NO_DUPLICATE_KEY;
         EDITS_MADE = TRUE;
      }
   }
   else   {
      beep();
      prompt ("No Defoliation Years to Delete!");
   }
   return (0);
} /* remove_all_defol ****************************************************/


void initialize_defol_yr_struct ( DEFOL_YRS *struct_ptr )
/*
DESCRIPTION:
  initialize DEFOL_YRS defoliation structure
CALLED BY:
  add_defol_yr
SAMPLE CALL
  initialize_defol_yr_struct ( &defol_yr_struct );
*/
{
   int i;

   for (i = 0; i < MAX_HOSTS; i++)   {
      struct_ptr->defol_amt[0][i] = 0;
      struct_ptr->defol_amt[1][i] = 0;
   }
   struct_ptr->next = NULL_DEFOL_PTR;
}  /*  initialize_defol_yr_struct ***************************************************/
```

```
/******** DEFOLIATION LINKED LIST VARIABLES HEADER FILE ************/
/* not included in defines.h because of possible conflict with CSCAPE boolean */
#define FALSE                 0
#define TRUE                  1

/* DEFOLIATION data file variables used with linked list */
/* structure for one defoliation year */
typedef struct defol_str {
   int defol_yr_key;              /* defoliation yr - used as key */
   int defol_amt[2][MAX_HOSTS];   /* defoliation % ;understory & overstory by host */
   struct defol_str *next;        /* ptr to next structure in list */
} DEFOL_YRS;


/* FUNCTION PROTOTYPES */
DEFOL_YRS *linked_defol_yr_insert (DEFOL_YRS *list_start, DEFOL_YRS new_str );
void linked_defol_yr_extract (DEFOL_YRS *list_start, key_type search_key,
                         DEFOL_YRS *extracted_str, int *LOCATED );
DEFOL_YRS *linked_defol_yr_delete (DEFOL_YRS *list_start, key_type old_key,
                              int *status );
void linked_defol_yr_duplication (DEFOL_YRS *list_start, int key, int *status );
```

```
/*************** DEFOLIATION LINKED LIST ROUTINES  *****************/
/* Microsoft C 6.0 includes */
#include <stdio.h>
#include <malloc.h> /* large memory model: returns 4 byte ptrs */
#include "defines.h"
#include "dlink.h"

DEFOL_YRS *linked_defol_yr_insert (DEFOL_YRS *list_start,
                                   DEFOL_YRS new_str )
/*
DESCRIPTION:
  inserts structure into linked list, keeping list in ascending order
ARGUMENTS:
  list_start (in\out) - ptr to front of list
  new_str    (in) - structure to insert in the list
RETURNS:
  list_start - ptr to DEFOL_YRS
SAMPLE CALL:
  start_defol_yr_ptr = linked_defol_yr_insert (start_defol_yr_ptr, new_str );
CALLED BY:
  add_defol_yr, read_defol
NOTES:
  initialize list_start to NULL_DEFOL_PTR before calling routine the 1st time
*/
{
   DEFOL_YRS *search_ptr;       /* ptr used for searching */
   DEFOL_YRS *last_ptr;         /* the last structure searched */
   DEFOL_YRS *temp_ptr;         /* temp structure */
   int LOCATED;                 /* used as boolean */

   /* create new structure; place info in it */
   temp_ptr = (DEFOL_YRS *)malloc (sizeof (DEFOL_YRS));

   if (temp_ptr == NULL_DEFOL_PTR)  /* malloc returns NULL if space not available */
      printf ("Couldn't allocate space\n");
   else
   {
      *temp_ptr = new_str;
      temp_ptr->next = NULL_DEFOL_PTR;

      search_ptr = list_start;

      /* if the list is empty, make the new record the only one */
      if (search_ptr == NULL_DEFOL_PTR)
         list_start = temp_ptr;
      else
      {
         /* search until just before list is exhausted */
         LOCATED = FALSE;
         while ((search_ptr != NULL_DEFOL_PTR) && (LOCATED == FALSE))
         {
            if (search_ptr->defol_yr_key < new_str.defol_yr_key)
            {
               last_ptr = search_ptr;
               search_ptr = search_ptr->next;
            }
            else
               LOCATED = TRUE;
         }

         /* have the new structure point to the next structure */
         temp_ptr->next = search_ptr;

         /* if new str will be the 1st str, set starting ptr */
         if (search_ptr == list_start)
            list_start = temp_ptr;
         else /* point last str to new str */
            last_ptr->next = temp_ptr;
      }
   }
   return (list_start);
}  /* linked_defol_yr_insert ************************************************/


void linked_defol_yr_duplication (DEFOL_YRS *list_start,
                                  int key,
                                  int *status )
/*
DESCRIPTION:
  checks for duplication of key insertion into linked list
ARGUMENTS:
  list_start (in\out) - ptr to front of list
```

```
   key (in) - key to test for duplication in linked list
   status (out) - error status  (global but passed anyway)
 SAMPLE CALL:
   linked_defol_yr_duplication (start_defol_yr_ptr, 1980, &status );
 CALLED BY:
   add_defol_yr
 */
 {
    DEFOL_YRS *search_ptr;  /* ptr used for searching */
    int LOCATED_DUPLICATE;          /* used as boolean */

    *status = NO_DUPLICATE_KEY;
    search_ptr = list_start;

    /* search until just before list is exhausted */
    LOCATED_DUPLICATE = FALSE;
    while ((search_ptr != NULL_DEFOL_PTR) && (LOCATED_DUPLICATE == FALSE))
    {
       if (search_ptr->defol_yr_key == key)
       {
          LOCATED_DUPLICATE = TRUE;
          *status = DUPLICATE_KEY;
       }
       else
          search_ptr = search_ptr->next;
    }
 } /* linked_defol_yr_duplication ************************************************/


 void linked_defol_yr_extract (DEFOL_YRS *list_start,
                               key_type search_key,
                               DEFOL_YRS *extracted_str,
                               int *LOCATED )
 /*
 DESCRIPTION:
   extracts structure from linked list; if keyed entry not found, return error
 ARGUMENTS:
   list_start (in) - ptr to front of list
   search_key (in) - keyed entry to search for
   *extracted_str (out) - entire structure found
   *LOCATED (out) - error flag (TRUE if structure found)
 SAMPLE CALL:
   linked_defol_yr_extract (start_defol_yr_ptr, search_key, &str, &LOCATED);
 CALLED BY:
   delete_defol_yr
 */
 {
    DEFOL_YRS *search_ptr;  /* ptr used for searching */

    /* search at the top of the list */
    search_ptr = list_start;

    /* search until key is found, or until list exhausted */
    *LOCATED = FALSE;
    while ((search_ptr != NULL_DEFOL_PTR) && (*LOCATED == FALSE))
    {
       if (search_ptr->defol_yr_key != search_key)
          search_ptr = search_ptr->next;
       else
          *LOCATED = TRUE;
    }
    if (*LOCATED == TRUE)
       *extracted_str = *search_ptr;
 }  /* linked_defol_yr_extract ***************************************************/


 DEFOL_YRS *linked_defol_yr_delete (DEFOL_YRS *list_start,
                                    key_type old_key,
                                    int *status )
 /*
 DESCRIPTION:
   deletes structure from linked list;
   if list empty or if key not found, error returned
 ARGUMENTS:
   list_start (in\out) - ptr to front of list
   old_key     (in) - key entry of structure to delete
   status      (out) - error status  (global but passed anyway)
 RETURNS:
   list_start - ptr to DEFOL_YRS
 SAMPLE CALL:
   start_defol_yr_ptr = linked_defol_yr_delete (start_defol_yr_ptr, 1980, &status);
 CALLED BY:
```

```
  delete_defol_yr
NOTES:
  see #define definitions for status codes
*/
{
   DEFOL_YRS *search_ptr;  /* ptr for searching */
   DEFOL_YRS *temp_ptr;    /* temp structure */
   int LOCATED;           /* used as boolean - was the location found yet */

   /* if the list is empty, set the error flag */
   if (list_start == NULL_DEFOL_PTR)
      *status = LIST_EMPTY;

   /* if structure to delete is 1st str, move starting ptr */
   else if (old_key == list_start->defol_yr_key)
   {
      /* use temp_ptr to dispose unwanted str */
      temp_ptr = list_start;
      list_start = list_start->next;
      free (temp_ptr);
      *status = SUCCESSFUL_DELETION;
   }
   else
   {
      /* search until list is exhausted */
      LOCATED = FALSE;
      search_ptr = list_start;
      while ((search_ptr->next != NULL_DEFOL_PTR) && (LOCATED == FALSE))
      {
         if (search_ptr->next->defol_yr_key != old_key)
            search_ptr = search_ptr->next;
         else
            LOCATED = TRUE;
      }

      /* if key not found, set error flag */
      if (LOCATED == FALSE)
         *status = KEY_NOT_FOUND;
      else
      {
         /* use temp_ptr to get rid of unwanted str */
         temp_ptr = search_ptr->next;

         /* close the list */
         search_ptr->next = search_ptr->next->next;

         free(temp_ptr);
         *status = SUCCESSFUL_DELETION;
      }
   }
   return (list_start);
}  /* linked_defol_yr_delete ************************************************/
```

```
/***************** STAND MANAGEMENT VARIABLES & FUNCTIONS HEADER FILE *************/
/* CSCAPE window variables */
extern int win_width, /* window width */
    win_x,            /* window position (x coord) of upper left hand corner */
    win_y,            /* window position (y coord) of upper left hand corner */
    win_height;       /* window height */


/* GENERAL data file variables */
extern int NYEAR,
    NHOSTS,
    ISYEAR;

extern MGMT_YRS *start_mgmt_yr_ptr, mgmt_yr_struct;

extern TREE tree[MAX_HOSTS];  /* array of tree structures */

extern int LOCATED,
   status,          /* == SUCCESSFUL_DELETION, KEY_NOT_FOUND, LIST_EMPTY, etc. */
   number_of_mgmt_yrs;     /* number of Management Years entered */


/* FILE ACCESS variables */
extern int file_options;


/* USER INTERFACE variables */
extern boolean EDITS_MADE,
   CHOICE;
extern int units;


/* USER INTERFACE HOST SPECIES variables */
extern char *tree_strings[];
/*****************************************************************************/
```

```
/********************** STAND MANAGEMENT ROUTINES ************************/
/* Microsoft C 6.0 includes */
#include <stdio.h>
#include <stdlib.h>
/* CSCAPE 3.2 includes */
#include <cscape.h>
#include <popdecl.h>
/* define file */
#include "defines.h"
#include "struct.h"
#include "mgmt.h"

/* FUNCTION PROTOTYPES */
/* routines using CSCAPE function calls */
void verify_choice ( char msg1[50], char msg2[50] );
int choose_mgmt_yr_edit (VOID *sdata, int idata);
int add_mgmt_yr (VOID *sdata, int idata);
int delete_mgmt_yr (VOID *sdata, int idata);
PRIVATE void mgmt_edit_for_one_yr(MGMT_YRS *struct_ptr, int display_only, int old_yr);
int change_mgmt(VOID *sdata, int idata);
int change_one_mgmt_yr (int old_yr);

/* no CSCAPE */
PRIVATE void initialize_mgmt_yr_struct ( MGMT_YRS *struct_ptr );
MGMT_YRS *linked_mgmt_yr_insert (MGMT_YRS *list_start, MGMT_YRS new_str );
void linked_mgmt_yr_extract (MGMT_YRS *list_start, key_type search_key,
                        MGMT_YRS *extracted_str, int *LOCATED );
MGMT_YRS *linked_mgmt_yr_delete (MGMT_YRS *list_start, key_type old_key,
                            int *status );
void linked_mgmt_yr_duplication (MGMT_YRS *list_start, int key, int *status );
int get_tree_index ( char tname[3] );


/***************************************/
int add_mgmt_yr (VOID *sdata, int idata)
/***************************************
DESCRIPTION:
  adds a management year
CALLED BY:
  framer menu ADD MANAGEMENT YEAR
CALLS:
  initialize_mgmt_yr_structure, linked_mgmt_yr_duplication, mgmt_edit_for_one_yr,
  linked_mgmt_yr_insert
NOTES:
  MAX_MGMT_YRS is maximum allowed
*/
{
   menu_type menu;
   sed_type add_yr_sed;
   char range_string[25];   /* e.g. "(1978,2000)\n"  */
   char start_yr_string[5];      /*  "1990\n" */
   char end_yr_string[5];
   int ending_yr;

   if (number_of_mgmt_yrs < MAX_MGMT_YRS )  {
      mgmt_yr_struct.mgmt_yr_key = 0;
      mgmt_yr_struct.manage = 0;
      ending_yr = ISYEAR + NYEAR - 1;
      sprintf (range_string, "(0,0)(%d,%d)", ISYEAR, ending_yr);
      sprintf (start_yr_string, "%d", ISYEAR);
      sprintf (end_yr_string, "%d", ending_yr);
      if ((menu = menu_Open()) == NULL)  {
         printf ("unable to open menu");
         return(0);
      }
      win_width = 66;
      win_y = 7;
      win_x = 5;
      menu_Printf(menu, "Starting Year: @a[0x1F]%s@a[0x17] ",
         start_yr_string);
      menu_Printf(menu, "  Ending Year: @a[0x1F]%s@a[0x17] \n",
         end_yr_string);
      menu_Printf(menu, "Management Year: @fd2[#####]\n",
         &mgmt_yr_struct.mgmt_yr_key, &int_funcs,"Year to manage stand data; MYEAR", range_string);
      menu_Printf(menu, "Management Method: @fd2[###]\n",
         &mgmt_yr_struct.manage, &int_funcs,"1:Proportion of Stems Removed 2:Target Residual Stem Count
(MANAGE)", "(0,2)");
      menu_Flush(menu);
      if ((add_yr_sed = sed_Open(menu)) == NULL)  {
         printf ("unable to open sed");
         return(0);
      }
```

```
      sed_SetLabel (add_yr_sed, 33);
      sed_SetWidth(add_yr_sed, win_width);
      sed_SetPosition(add_yr_sed, win_y, win_x);
      sed_SetColors (add_yr_sed, CYAN_BLUE, WHITE_BLUE, YELLOW_CYAN);
      sed_SetShadow(add_yr_sed, 1);
      sed_SetShadowAttr(add_yr_sed, DGRAY_BLACK);
      sed_SetMouse(add_yr_sed, sedmou_GreedyTrack);
      sed_SetBorder(add_yr_sed, bd_std);
      sed_SetBorderTitle(add_yr_sed, "            ADD A MANAGEMENT YEAR");
      sed_Repaint(add_yr_sed);
      sed_Go(add_yr_sed);
      /* year entered? */
      if (mgmt_yr_struct.mgmt_yr_key > 0)  {
          if (mgmt_yr_struct.manage > 0)  {
             /* does duplicate year exist? */
             linked_mgmt_yr_duplication (start_mgmt_yr_ptr, mgmt_yr_struct.mgmt_yr_key, &status );
             if (status == NO_DUPLICATE_KEY)  {
                initialize_mgmt_yr_struct ( &mgmt_yr_struct );
                mgmt_edit_for_one_yr(&mgmt_yr_struct, FALSE, 0);
                /* insert the edited structure into the linked_list */
                start_mgmt_yr_ptr = linked_mgmt_yr_insert (start_mgmt_yr_ptr, mgmt_yr_struct );
                number_of_mgmt_yrs = number_of_mgmt_yrs + 1;
             }
             else  { /* status == DUPLICATE_KEY */
                beep();
                prompt (" Duplicate management year entered");
             }
          }
          else  {
             beep();
             prompt (" Method should be 1 or 2.");
          }
       }
       sed_Close(add_yr_sed);
    }
    else   {
      beep();
      pop_Prompt(" No more management years to enter! (15 maximum)", -1, -1, -1, 35, WHITE_RED, bd_2);
    }
    return (0);
} /* add_mgmt_yr ************************************************************/


/*****************************************/
void mgmt_edit_for_one_yr (MGMT_YRS *struct_ptr, int display_only, int old_yr)
/*****************************************
DESCRIPTION:
   Stand management parameter editing for one management year
*/
{
   int i;
   menu_type menu;
   sed_type mgmt_edit_sed;
   char yr_string[5];   /* e.g. "1990\n" */

   if (units == SI)
      for (i = 0; i < NHOSTS; i++)  {
         struct_ptr->cutmin[i] *= IN_to_CM;
         struct_ptr->cutmax[i] *= IN_to_CM;
         if (struct_ptr->manage == TARGETED)
            struct_ptr->action[i] *= AC_to_HA;
      }
   EDITS_MADE = TRUE;
   file_options = file_options | STDMAN;
   sprintf (yr_string, "%d", struct_ptr->mgmt_yr_key);
   if ((menu = menu_Open()) == NULL)
      printf ("unable to open menu");
   win_width = 56;
   win_y = 12;
   if (display_only)
      win_x = 4;
   else
      win_x = 16;
   if (struct_ptr->manage == TARGETED)   {
      menu_Printf(menu, " YEAR: @a[0x1F]%s@a[0x17]      Target Residual Stem Count\n",
         yr_string);
      menu_Printf(menu, "       MIN      MAX    TARGET\n");
   }
   else   {
      menu_Printf(menu, " YEAR: @a[0x1F]%s@a[0x17]       Proportion of Stems Removed\n",
         yr_string);
      menu_Printf(menu, "       MIN      MAX    PROPORTION\n");
```

```
   }
   for (i = 0; i < NHOSTS; i++)   {
     menu_Printf(menu, " %s ", tree[i].TNAME);
     menu_Printf(menu,"   @fd2[@[8,#]]",
       &struct_ptr->cutmin[i], &sfloat_funcs,"Minimum Diameter; 0->200; CUTMIN", "(0,200)");
     menu_Printf(menu," @fd2[@[8,#]]",
       &struct_ptr->cutmax[i], &sfloat_funcs,"Maximum Diameter; 0->200; CUTMAX", "(0,200)");
     if (struct_ptr->manage == TARGETED)
        menu_Printf(menu," @fd2[@[8,#]]",
          &struct_ptr->action[i], &sfloat_funcs,"Target; > 0", "(0,)");
     else /* struct_ptr->manage == proportion */
        menu_Printf(menu,"   @fd2[#####]",
          &struct_ptr->action[i], &sfloat_funcs,"Proportion; 0->1", "(0,1)");
     menu_Printf(menu, "    %s%s%s\n","(", tree_strings[get_tree_index(tree[i].TNAME)], ")");
   }
   menu_Flush(menu);
   if ((mgmt_edit_sed = sed_Open(menu)) == NULL)
      printf ("unable to open sed");
   sed_SetLabel (mgmt_edit_sed, 34);
   sed_SetWidth(mgmt_edit_sed, win_width);
   sed_SetPosition(mgmt_edit_sed, win_y, win_x);
   sed_SetColors (mgmt_edit_sed, CYAN_BLUE, WHITE_BLUE, YELLOW_CYAN);
   sed_SetShadow(mgmt_edit_sed, 1);
   sed_SetShadowAttr(mgmt_edit_sed, DGRAY_BLACK);
   sed_SetMouse(mgmt_edit_sed, sedmou_GreedyTrack);
   sed_SetBorder(mgmt_edit_sed, bd_std);
   sed_SetBorderTitle(mgmt_edit_sed, "                STAND MANAGEMENT");
   sed_SetSpecial(mgmt_edit_sed, inter_field_grid);
   sed_Repaint(mgmt_edit_sed);
   if (display_only)
      (void)change_one_mgmt_yr(old_yr);
   else
      sed_Go(mgmt_edit_sed);
   sed_Close(mgmt_edit_sed);
   if (units == SI)
      for (i = 0; i < NHOSTS; i++)  {
         struct_ptr->cutmin[i] /= IN_to_CM;
         struct_ptr->cutmax[i] /= IN_to_CM;
         if (struct_ptr->manage == TARGETED)
            struct_ptr->action[i] /= AC_to_HA;
      }
} /* mgmt_edit_for_one_yr ************************************************/


int delete_mgmt_yr (VOID *sdata, int idata)
/*
DESCRIPTION:
  displays available management years; Deletes year chosen
CALLED BY:
  framer menu DELETE MANAGEMENT YEAR
CALLS:
  verify_choice, linked_mgmt_yr_extract, linked_mgmt_yr_delete
*/
{
   menu_type menu;
   sed_type del_mgmt_sed;
   char yr_string[5];   /* e.g. "1990\n" */
   int yr_key_to_delete = 0;
   MGMT_YRS *search_ptr;

   search_ptr = start_mgmt_yr_ptr;
   if (search_ptr != NULL_MGMT_PTR)
   {
      if ((menu = menu_Open()) == NULL)
      {
         printf ("unable to open menu");
         return(0);
      }
      win_width = 35;
      win_y = 6;
      win_x = 5;
      if ((number_of_mgmt_yrs+1) < (disp_GetHeight()-10))
         win_height = number_of_mgmt_yrs+1;
      else
         win_height = disp_GetHeight()-10;
      while (search_ptr != NULL_MGMT_PTR)
      {
         sprintf (yr_string, "%d", search_ptr->mgmt_yr_key);
         menu_Printf(menu, " %s %s",
           yr_string, "\n");
```

```
            search_ptr = search_ptr->next;
         }
         menu_Printf(menu, "Delete Year:@fd[#####]",
            &yr_key_to_delete, &int_funcs,"Year to delete");
         menu_Flush(menu);
         if ((del_mgmt_sed = sed_Open(menu)) == NULL)
         {
            printf ("unable to open sed");
            return(0);
         }
         sed_SetLabel (del_mgmt_sed, 35);
         sed_SetWidth(del_mgmt_sed, win_width);
         sed_SetPosition(del_mgmt_sed, win_y, win_x);
         sed_SetColors (del_mgmt_sed, WHITE_RED, WHITE_BLUE, RED_WHITE);
         sed_SetShadow(del_mgmt_sed, 1);
         sed_SetShadowAttr(del_mgmt_sed, DGRAY_BLACK);
         sed_SetMouse(del_mgmt_sed, sedmou_GreedyClick);
         sed_SetBorder(del_mgmt_sed, bd_std);
         sed_SetBorderTitle(del_mgmt_sed, "CHOOSE MANAGEMENT YEAR TO DELETE");
         sed_Repaint(del_mgmt_sed);
         sed_Go(del_mgmt_sed);
         if (yr_key_to_delete > 0)
         {
            /*initialized so struct can be sent to extract to check for existence*/
            mgmt_yr_struct.mgmt_yr_key = 1;
            mgmt_yr_struct.manage = 1;
            initialize_mgmt_yr_struct ( &mgmt_yr_struct );
            linked_mgmt_yr_extract (start_mgmt_yr_ptr, yr_key_to_delete, &mgmt_yr_struct,
                                    &LOCATED);
            if (LOCATED == FALSE)
            {
               beep();
               prompt ("management year not found in list");
            }
            else  /* LOCATED */
            {
               CHOICE = FALSE;
               verify_choice ( "Delete indicated Managment Year?",
                               "WARNING: Management Year will be lost");
               if (CHOICE == TRUE)
               {
                  EDITS_MADE = TRUE;
                  file_options = file_options | STDMAN;
                  start_mgmt_yr_ptr = linked_mgmt_yr_delete (start_mgmt_yr_ptr, yr_key_to_delete,
                                                             &status);
                  if (status == KEY_NOT_FOUND)
                     prompt ("management year not found in list");
                  else
                     number_of_mgmt_yrs = number_of_mgmt_yrs - 1;
                  /* if no more mgmt yrs, reset STDMAN reset bit to 0 */
                  if (0 == number_of_mgmt_yrs)
                     if ((file_options & STDMAN) == STDMAN) /* is STDMAN set? */
                        file_options = STDMAN ^ file_options; /* bitwise XOR */
               }
            }
         }
         sed_Close(del_mgmt_sed);
      }
      else
      {
         beep();
         prompt ("No Management Years to Delete!");
      }
      return (0);
} /* delete_mgmt_yr ********************************************************/


int choose_mgmt_yr_edit (VOID *sdata, int idata)
/*
DESCRIPTION:
  Displays available management years to edit; Select from list with space bar
CALLED BY:
  framer menu EDIT MANAGEMENT YEAR
CALLS:
  mgmt_edit_for_one_yr
*/
{
   int i, j;
   menu_type menu;
   sed_type choose_sed;
   boolean SELECTED[MAX_MGMT_YRS];
   char yr_string[5];   /* e.g. "1990\n" */
```

```
   MGMT_YRS *search_ptr;

   search_ptr = start_mgmt_yr_ptr;
   if (search_ptr != NULL_MGMT_PTR)
   {
      if ((menu = menu_Open()) == NULL)
      {
         printf ("unable to open menu");
         return(0);
      }
      win_width = 45;
      win_y = 4;
      win_x = 5;
      if ((number_of_mgmt_yrs) < (disp_GetHeight()-10))
         win_height = number_of_mgmt_yrs;
      else
         win_height = disp_GetHeight()-10;
      for (j = 0; j < MAX_MGMT_YRS; j++)
         SELECTED[j] = FALSE;
      i = 0;
      while (search_ptr != NULL_MGMT_PTR)
      {
         sprintf (yr_string, "%d", search_ptr->mgmt_yr_key);
         menu_Printf(menu, "@f[# %s]\n", &SELECTED[i], &check_funcs,
           yr_string );
         search_ptr = search_ptr->next;
         i = i + 1;
      }
      menu_Flush(menu);
      if ((choose_sed = sed_Open(menu)) == NULL)
      {
         printf ("unable to open sed");
         return(0);
      }
      sed_SetLabel (choose_sed, 36);
      sed_SetWidth(choose_sed, win_width);
      sed_SetHeight(choose_sed, win_height);
      sed_SetPosition(choose_sed, win_y, win_x);
      sed_SetColors (choose_sed, CYAN_BLUE, WHITE_BLUE, YELLOW_BLACK);
      sed_SetShadow(choose_sed, 1);
      sed_SetShadowAttr(choose_sed, DGRAY_BLACK);
      sed_SetMouse(choose_sed, sedmou_GreedyTrack);
      sed_SetBorder(choose_sed, bd_title);
      sed_SetBorderTitle(choose_sed, "CHOOSE MANAGEMENT YEARS TO EDIT (use SPACE bar)");
      sed_Repaint(choose_sed);
      sed_Go(choose_sed);

      search_ptr = start_mgmt_yr_ptr;
      i = 0;
      while (search_ptr != NULL_MGMT_PTR)
      {
         if (SELECTED[i] != FALSE)
                  mgmt_edit_for_one_yr(search_ptr, FALSE, 0);
         search_ptr = search_ptr->next;
         i = i + 1;
      }
      sed_Close(choose_sed);
   }
   else
   {
      beep();
      prompt (" Add a management year first.");
   }
   search_ptr = start_mgmt_yr_ptr;
   return (0);
} /* choose_mgmt_yr_edit ************************************************************/


/****************************************/
int change_mgmt(VOID *sdata, int idata)
/****************************************
DESCRIPTION: modify the management year, leaving the data intact
*/
{
   int i, j, ret;
   menu_type menu;
   sed_type sed;
   char yr_string[17];
   MGMT_YRS *search_ptr;
   int yr_holder[MAX_MGMT_YRS];
   int yr_index = 0;
```

```
    search_ptr = start_mgmt_yr_ptr;
    if (search_ptr != NULL_MGMT_PTR) {
       if ((menu = menu_Open()) == NULL)  {
          printf ("unable to open menu");
          return(0);
       }
       win_width = 23;
       win_y  = 4;
       win_x  = 15;
       if ((number_of_mgmt_yrs) < (disp_GetHeight()-10))
          win_height = number_of_mgmt_yrs;
       else
          win_height = disp_GetHeight()-10;
       while (search_ptr != NULL_MGMT_PTR)  {
          sprintf(yr_string, " @f[ %d ]\n", search_ptr->mgmt_yr_key);
          menu_Printf(menu, yr_string, NULL, &menu_funcs);
          yr_holder[yr_index++] = search_ptr->mgmt_yr_key;
          search_ptr = search_ptr->next;
       }
       menu_Flush(menu);
       if ((sed = sed_Open(menu)) == NULL)  {
          printf ("unable to open sed");
          return(0);
       }
       sed_SetLabel(sed, 32);
       sed_SetWidth(sed, win_width);
       sed_SetHeight(sed, win_height);
       sed_SetPosition(sed, win_y, win_x);
       sed_SetColors (sed, CYAN_BLUE, WHITE_BLUE, YELLOW_CYAN);
       sed_SetShadow(sed, 1);
       sed_SetShadowAttr(sed, DGRAY_BLACK);
       sed_SetMouse(sed, sedmou_GreedyTrack);
       sed_SetBorder(sed, bd_title);
       sed_SetBorderTitle(sed, "   Change Which Year? ");
       sed_Repaint(sed);
       ret = sed_Go(sed);
       if (ret > 0)  {
          linked_mgmt_yr_extract (start_mgmt_yr_ptr, yr_holder[ret-1], &mgmt_yr_struct,&LOCATED);
            mgmt_edit_for_one_yr(&mgmt_yr_struct, TRUE, yr_holder[ret-1]);
       }
       sed_Close(sed);
    }
    else   {
       beep();
       prompt (" Add a management year first.");
    }
    return (0);
}


/*************************************/
int change_one_mgmt_yr (int old_yr)
/*************************************
DESCRIPTION: modify the year, leaving the data intact
*/
{
   menu_type menu;
   sed_type sed;
   MGMT_YRS *search_ptr;
   int available_yrs = 0;
   int year, new_yr = 0, ret;
   int year_index = 0;
   char yr_string[5];

   if ((menu = menu_Open()) == NULL)  {
      printf ("unable to open menu");
      return(0);
   }
   win_width = 25;
   win_y = 7;
   win_x = 46;
   sprintf (yr_string, "%d", old_yr);
   menu_Printf(menu, "   Old Year: @a[0x1F]%s@a[0x17]\n", yr_string);
   for (year = ISYEAR; year < ISYEAR+NYEAR; year++)
      if (year != old_yr) {
         linked_mgmt_yr_duplication(start_mgmt_yr_ptr, year, &status);
         if(status == NO_DUPLICATE_KEY)  {
                menu_Printf(menu, " @f[ %d ]\n", NULL, &menu_funcs, year);
            available_yrs++;
         }
      }
   if (available_yrs == 0) {
```

```
      menu_Destroy(menu);
      prompt(" No available years to change to");
      return(-1);
   }
   menu_Flush(menu);
   if ((sed = sed_Open(menu)) == NULL) {
      printf ("unable to open sed");
      return(0);
   }
   sed_SetLabel (sed, 62);
   sed_SetWidth(sed, win_width);
   sed_SetPosition(sed, win_y, win_x);
   sed_SetHeight(sed, min(available_yrs+1, disp_GetHeight()-8));
   sed_SetColors (sed, CYAN_BLUE, WHITE_BLUE, YELLOW_CYAN);
   sed_SetShadow(sed, 1);
   sed_SetShadowAttr(sed, DGRAY_BLACK);
   sed_SetMouse(sed, sedmou_GreedyTrack);
   sed_SetBorder(sed, bd_mouse);
   sed_SetBorderTitle(sed, "Choose year to change to");
   sed_Repaint(sed);
   ret = sed_Go(sed);
   if (ret > 0) {
      for (year = ISYEAR; year < ISYEAR+NYEAR; year++)
         if (year != old_yr) {
            /* is this year already in list? */
            linked_mgmt_yr_duplication(start_mgmt_yr_ptr, year, &status);
            if (status == NO_DUPLICATE_KEY)  {
               year_index++;
               if (ret == year_index)
                  new_yr = year;
         }
      }
      if (new_yr > 0)  {
         /* get structure we need to move */
         linked_mgmt_yr_extract (start_mgmt_yr_ptr, old_yr, &mgmt_yr_struct,&LOCATED);
         if (LOCATED == FALSE)
            prompt ("ERROR: Management year not found");

         /* delete old node */
         start_mgmt_yr_ptr = linked_mgmt_yr_delete(start_mgmt_yr_ptr, old_yr,&status);
         if(status == KEY_NOT_FOUND)
            prompt ("ERROR: Cannot delete management year");

         /* insert structure (updated with new year) */
         mgmt_yr_struct.mgmt_yr_key = new_yr;
         start_mgmt_yr_ptr = linked_mgmt_yr_insert(start_mgmt_yr_ptr, mgmt_yr_struct);
      }
   }
   sed_Close(sed);
   return (ret);
}


void initialize_mgmt_yr_struct ( MGMT_YRS *struct_ptr )
/*
DESCRIPTION:
  initialize MGMT_YRS management structure
CALLED BY:
  add_mgmt_yr
SAMPLE CALL
  initialize_mgmt_yr_struct ( &mgmt_yr_struct );
*/
{
   int i;

   for (i = 0; i < MAX_HOSTS; i++)
   {
      struct_ptr->cutmin[i] = 2.0F;
      struct_ptr->cutmax[i] = 20.0F;
      if (struct_ptr->manage == TARGETED)
         struct_ptr->action[i] = 50.0F;
      else /* proportion */
         struct_ptr->action[i] = 0.5F;
   }
   struct_ptr->next = NULL_MGMT_PTR;
}  /*  initialize_mgmt_yr_struct ************************************************/
```

```
/******** STAND MANAGEMENT LINKED LIST VARIABLES & FUNCTIONS HEADER FILE *************/
/* not included in defines.h because of possible conflict with CSCAPE boolean */
#define FALSE                0
#define TRUE                 1

/* structure for one management year */
typedef struct mgmt_str {
   int mgmt_yr_key;         /*FORTRAN's MYEARS - used as key for linked list */
   int manage;              /*Management Method 1-Target Stem Count 2-Proportion of Stems */
   float cutmin[MAX_HOSTS];
   float cutmax[MAX_HOSTS];
   float action[MAX_HOSTS];  /* action may be target or proportion depending on manage */
   struct mgmt_str *next;    /* ptr to the next structure in the linked list */
} MGMT_YRS;


/* FUNCTION PROTOTYPES */
MGMT_YRS *linked_mgmt_yr_insert (MGMT_YRS *list_start, MGMT_YRS new_str );
void linked_mgmt_yr_extract (MGMT_YRS *list_start, key_type search_key,
                        MGMT_YRS *extracted_str, int *LOCATED );
MGMT_YRS *linked_mgmt_yr_delete (MGMT_YRS *list_start, key_type old_key,
                           int *status );
void linked_mgmt_yr_duplication (MGMT_YRS *list_start, int key, int *status );
```

```
/*************** STAND MANAGEMENT LINKED LIST ROUTINES  ****************/
/* Microsoft C 6.0 includes */
#include <stdio.h>
#include <malloc.h> /* large memory model: returns 4 byte ptrs */
/* define file */
#include "defines.h"
/* variables and functions for stand management linked list routines */
#include "mlink.h"


/* linked list routines converted from Pascal to C
   from "TURBO PASCAL Programmer's library"  */
MGMT_YRS *linked_mgmt_yr_insert (MGMT_YRS *list_start,
                                 MGMT_YRS new_str )
/*
DESCRIPTION:
  inserts structure into linked list, keeping list in ascending order
ARGUMENTS:
  list_start (in\out) - ptr to front of list
  new_str    (in) - structure to insert in the list
RETURNS:
  list_start - ptr to MGMT_YRS
SAMPLE CALL:
  start_mgmt_yr_ptr = linked_mgmt_yr_insert (start_mgmt_yr_ptr, new_str );
CALLED BY:
  add_mgmt_yr, read_mgmt
NOTES:
  initialize list_start to NULL_MGMT_PTR before calling routine the 1st time
*/
{
   MGMT_YRS *search_ptr;      /* ptr used for searching */
   MGMT_YRS *last_ptr;        /* the last structure searched */
   MGMT_YRS *temp_ptr;   /* temp structure */
   int LOCATED;               /* used as boolean */

   /* create new structure; place info in it */
   temp_ptr = (MGMT_YRS *)malloc (sizeof (MGMT_YRS));

   if (temp_ptr == NULL_MGMT_PTR)  /* malloc returns NULL if space not available */
      printf ("Couldn't allocate space\n");
   else
   {
      *temp_ptr = new_str;
      temp_ptr->next = NULL_MGMT_PTR;

      search_ptr = list_start;

      /* if the list is empty, make the new record the only one */
      if (search_ptr == NULL_MGMT_PTR)
         list_start = temp_ptr;
      else
      {
         /* search until just before list is exhausted */
         LOCATED = FALSE;
         while ((search_ptr != NULL_MGMT_PTR) && (LOCATED == FALSE))
         {
            if (search_ptr->mgmt_yr_key < new_str.mgmt_yr_key)
            {
               last_ptr = search_ptr;
               search_ptr = search_ptr->next;
            }
            else
               LOCATED = TRUE;
         }

         /* have the new structure point to the next structure */
         temp_ptr->next = search_ptr;

         /* if new str will be the 1st str, set starting ptr */
         if (search_ptr == list_start)
            list_start = temp_ptr;
         else /* point last str to new str */
            last_ptr->next = temp_ptr;
      }
   }
   return (list_start);
}  /* linked_mgmt_yr_insert ************************************************/


void linked_mgmt_yr_duplication (MGMT_YRS *list_start,
                                 int key,
                                 int *status )
```

```
/*
DESCRIPTION:
  checks for duplication of key insertion into linked list
ARGUMENTS:
  list_start (in\out) - ptr to front of list
  key (in) - key to test for duplication in linked list
  status (out) - error status  (global but passed anyway)
SAMPLE CALL:
  linked_mgmt_yr_duplication (start_mgmt_yr_ptr, 1980, &status );
CALLED BY:
  add_mgmt_yr
*/
{
   MGMT_YRS *search_ptr;  /* ptr used for searching */
   int LOCATED_DUPLICATE;           /* used as boolean */

   *status = NO_DUPLICATE_KEY;
   search_ptr = list_start;

   /* search until just before list is exhausted */
   LOCATED_DUPLICATE = FALSE;
   while ((search_ptr != NULL_MGMT_PTR) && (LOCATED_DUPLICATE == FALSE))
   {
      if (search_ptr->mgmt_yr_key == key)
      {
         LOCATED_DUPLICATE = TRUE;
         *status = DUPLICATE_KEY;
      }
      else
         search_ptr = search_ptr->next;
   }
} /* linked_mgmt_yr_duplication ************************************************/


void linked_mgmt_yr_extract (MGMT_YRS *list_start,
                          key_type search_key,
                          MGMT_YRS *extracted_str,
                          int *LOCATED )
/*
DESCRIPTION:
  extracts structure from linked list; if keyed entry not found, return error
ARGUMENTS:
  list_start (in) - ptr to front of list
  search_key (in) - keyed entry to search for
  *extracted_str (out) - entire structure found
  *LOCATED (out) - error flag (TRUE if structure found)
SAMPLE CALL:
  linked_mgmt_yr_extract (start_mgmt_yr_ptr, search_key, &str, &LOCATED);
CALLED BY:
  delete_mgmt_yr
*/
{
   MGMT_YRS *search_ptr;  /* ptr used for searching */

   /* search at the top of the list */
   search_ptr = list_start;

   /* search until key is found, or until list exhausted */
   *LOCATED = FALSE;
   while ((search_ptr != NULL_MGMT_PTR) && (*LOCATED == FALSE))
   {
      if (search_ptr->mgmt_yr_key != search_key)
         search_ptr = search_ptr->next;
      else
         *LOCATED = TRUE;
   }
   if (*LOCATED == TRUE)
      *extracted_str = *search_ptr;
}  /* linked_mgmt_yr_extract *************************************************/


MGMT_YRS *linked_mgmt_yr_delete (MGMT_YRS *list_start,
                              key_type old_key,
                              int *status )
/*
DESCRIPTION:
  deletes structure from linked list;
  if list empty or if key not found, error returned
ARGUMENTS:
  list_start (in\out) - ptr to front of list
  old_key     (in) - key entry of structure to delete
  status      (out) - error status  (global but passed anyway)
```

```
RETURNS:
  list_start - ptr to MGMT_YRS
SAMPLE CALL:
  start_mgmt_yr_ptr = linked_mgmt_yr_delete (start_mgmt_yr_ptr, 1980, &status);
CALLED BY:
  delete_mgmt_yr
NOTES:
  see #define definitions for status codes
*/
{
   MGMT_YRS *search_ptr;  /* ptr for searching */
   MGMT_YRS *temp_ptr;    /* temp structure */
   int LOCATED;           /* used as boolean - was the location found yet */

   /* if the list is empty, set the error flag */
   if (list_start == NULL_MGMT_PTR)
      *status = LIST_EMPTY;

   /* if structure to delete is 1st str, move starting ptr */
   else if (old_key == list_start->mgmt_yr_key)
   {
      /* use temp_ptr to dispose unwanted str */
      temp_ptr = list_start;
      list_start = list_start->next;
      free (temp_ptr);
      *status = SUCCESSFUL_DELETION;
   }
   else
   {
      /* search until list is exhausted */
      LOCATED = FALSE;
      search_ptr = list_start;
      while ((search_ptr->next != NULL_MGMT_PTR) && (LOCATED == FALSE))
      {
         if (search_ptr->next->mgmt_yr_key != old_key)
            search_ptr = search_ptr->next;
         else
            LOCATED = TRUE;
      }

      /* if key not found, set error flag */
      if (LOCATED == FALSE)
         *status = KEY_NOT_FOUND;
      else
      {
         /* use temp_ptr to get rid of unwanted str */
         temp_ptr = search_ptr->next;

         /* close the list */
         search_ptr->next = search_ptr->next->next;

         free(temp_ptr);
         *status = SUCCESSFUL_DELETION;
      }
   }
   return (list_start);
}  /* linked_mgmt_yr_delete ************************************************/
```

```
/*************** WEATHER VARIABLES & FUNCTIONS HEADER FILE ************/
/* CSCAPE window variables */
extern int win_width, /* window width */
    win_x,            /* window position (x coord) of upper left hand corner */
    win_y,            /* window position (y coord) of upper left hand corner */
    win_height;       /* window height */

/* USER INTERFACE variables */
extern boolean EDITS_MADE;
extern int units;

/* FILE ACCESS variables */
extern int file_options;

/* GENERAL data file variables */
extern int ISYEAR,
    NYEAR;

/* WEATHER data file variables */
extern float
    degree_days[MAX_SIM_YRS];  /* total degree days for one year */


/* FUNCTION PROTOTYPES */
/* routines using CSCAPE function calls */
int weather_edit (VOID *sdata, int idata);
```

```
/********************* WEATHER ROUTINE ************************/
/* Microsoft C 6.0 includes */
#include <stdio.h>
/* CSCAPE 3.2 includes */
#include <cscape.h>
#include <popdecl.h>
/* define file */
#include "defines.h"
#include "weather.h"


int weather_edit (VOID *sdata, int idata)
/*
DESCRIPTION:
  edit annual degree days for all simulation years
CALLED BY:
  framer menu EDIT WEATHER DATA
*/
{
   int i, year;
   menu_type menu;
   sed_type sed;

   if (units == SI)
      for (i = 0; i < NYEAR; i++)
         degree_days[i] *= FDD_to_CDD;
   EDITS_MADE = TRUE;
   file_options = file_options | WEA;
   year = ISYEAR;
   if ((menu = menu_Open()) == NULL)  {
      printf ("unable to open menu");
      return(0);
   }
   win_width = 40;
   win_y = 8;
   win_x = 13;
   if ((NYEAR+1) < (disp_GetHeight()-9))
      win_height = NYEAR+1;
   else
      win_height = disp_GetHeight()-9;
   menu_Printf(menu, " YEAR       DEGREE-DAYS\n");
   for (i = 0; i < NYEAR; i++)  {
      menu_Printf(menu, " %d          ",
         year);
      if (units == ENGLISH)
         menu_Printf(menu, "@fd2[#######]\n",
                  &degree_days[i], &sfloat_funcs, "Total annual degree-days; 2000 -> 6000", "(2000,6000)");
      else
         menu_Printf(menu, "@fd2[#######]\n",
                  &degree_days[i], &sfloat_funcs, "Total annual degree-days; 1000 -> 3500", "(1000,3500)");
      year++;
   }
   menu_Flush(menu);
   if ((sed = sed_Open(menu)) == NULL)  {
      printf ("unable to open sed");
      return(0);
   }
   sed_SetLabel(sed, 29);
   sed_SetWidth(sed, win_width);
   sed_SetHeight(sed, win_height);
   sed_SetPosition(sed, win_y, win_x);
   sed_SetColors (sed, CYAN_BLUE, WHITE_BLUE, YELLOW_CYAN);
   sed_SetShadow(sed, 1);
   sed_SetShadowAttr(sed, DGRAY_BLACK);
   sed_SetMouse(sed, sedmou_GreedyTrack);
   sed_SetBorder(sed, bd_mouse);
   sed_SetBorderTitle(sed, "Weather Data");
   sed_Repaint(sed);
   sed_Go(sed);
   sed_Close(sed);
   if (units == SI)
      for (i = 0; i < NYEAR; i++)
         degree_days[i] /= FDD_to_CDD;
   return (0);
}/* weather_edit ***********************************************************/
```

```
/***************** MODEL OUTPUT VARIABLES & FUNCTIONS HEADER FILE **********/
#define S_IWRITE   0000200 /* (octal) write permission, owner; from <sys\stat.h> */

/*****************************************************************************/
/* PRINTER */
#define VIEW       0
#define PRINT      1

#define MSC    1          /* define this if using MS C */
/* #define TURBOC 1          define this if using turbo C */

/* port I/O functions */
#ifdef TURBOC
#include <dos.h>
#endif

#ifdef MSC
#include <conio.h>
#endif


/* define macros for BIOS call; Turbo C version */
#ifdef TURBOC
#define callprint(cmd,data,port)   \
                ( (unsigned) biosprint(cmd, data,port) )
#endif

/* Microsoft C version */
#ifdef MSC
#define callprint(cmd,data,port)   \
                ( _bios_printer(cmd,port,data) )
#endif

/* status bits */
#define INIT_OK                 0x00
#define NOT_BUSY_and_SELECTED   0x90
#define SELECTED                0x10


/* BIOS parallel printer command codes and return values */
#define PORT_LPT1     0
#define PRINT_CHAR    0     /* send a char to printer */
#define INIT_PTR      1     /* initialize the printer */
#define CANNOT_INIT   1
#define GET_STATUS    2     /* get printer status */
#define NOT_READY     2     /* returns busy or not selected */

/*****************************************************************************/
/* CSCAPE window variables */
extern int win_width, /* window width */
    win_x,            /* window position (x coord) of upper left hand corner */
    win_y,            /* window position (y coord) of upper left hand corner */
    win_height;       /* window height */

/*****************************************************************************/
/* USER INTERFACE variables */


extern boolean EDITS_MADE,
   CHOICE;
/*****************************************************************************/
/* FILE ACCESS variable */

extern int file_options;
/*****************************************************************************/
/* STAND data file variables */

extern int ICAT;
/*****************************************************************************/
/* GENERAL data file variables */

extern int NYEAR,
    IPYEAR,
    IGYEAR,
    ISYEAR,
    IVIEW[MAX_VIEWS];
/*****************************************************************************/
/* DAMAGE MODEL OUTPUT variables */


/* output files data structure */
struct output_file_str {
```

```
   char output_needed[2];      /* FORTRAN LOGICAL var; treated as 1 char string */
   boolean SAVE_OUTPUT;        /* TRUE if user wants to save output file */
   char description[31];       /* description of output that is needed */
   FILE *outputfile;           /* file created by damage model */
   char model_outputfile_name[13]; /* name of file created by damage model */
   char user_outputfile_name[13];  /* renamed file created by damage model */
};

/* array of structures */
extern struct output_file_str  outputs[MAX_OUTPUT_FILES];

char  model_graph_file_name[13];  /* from model; contains NHOSTS */
FILE *model_graph_file;
/*****************************************************************************/
/* FUNCTION PROTOTYPES */


/* routines using CSCAPE function calls */
void verify_choice ( char msg1[50], char msg2[50] );
int select_outputs(VOID *sdata, int idata);
int choose_view_or_print (VOID *sdata, int idata);
int output_to_save (VOID *sdata, int idata);
PRIVATE void create_outputfile_name(int file_index);
PRIVATE int IVIEW_edit (void);
PRIVATE void view_output (int file_index);
PRIVATE int print_output (int file_index);

/* no CSCAPE */
void remove_invalid_chars (char file_name[9]);
void initialize_output(void);
void delete_output_after_save(void);
PRIVATE void copy_file (char *from, char *to);
PRIVATE void delete_array_duplicates (int array[], int array_number);
PRIVATE void insertion_sort (int array[], int array_number);
PRIVATE void move_array_zeros (int array[], int array_number);
PRIVATE void exchange_values (int *ptr_A, int *ptr_B);
PRIVATE int printer_test(void);
PRIVATE void check_drive_letter (char file_name[9]);
PRIVATE int check_file_length(char file_name[13]);
/*****************************************************************************/
```

```
/********************** MODEL OUTPUT ROUTINES ************************/
/* Microsoft C 6.0 includes */
#include <stdio.h>
#include <string.h>     /* contains declarations for the string handling fncs */
#include <io.h>         /* for access */
/* for copy_file function */
#include <dos.h>           /* O_RDONLY */
#include <fcntl.h>         /* open */
#include <sys\types.h>     /* system level calls for file info */
/* CSCAPE 3.2 includes */
#include <cscape.h>
#include <popdecl.h>
/* define file */
#include "defines.h"
/* variables and functions for Model Output routines */
#include "output.h"

#define ERROR           1

/*****************************************************************************/
void copy_file (char *from, char *to)
/*
DESCRIPTION:
  copy one file to another
CALLED BY:
  output_to_save, print_output
*/
{
   int from_handle, to_handle, result;
   char buffer [512];

   /* open source file */
   if ((from_handle = open (from, O_RDONLY)) == -1)
   {
      perror ("\nError opening source file");
      return;
   }

   /* open destination file */
   if ((to_handle = creat (to, S_IWRITE)) == -1)
   {
      perror ("\nError opening target file");
      close (from_handle);
      return;
   }

   /* copy */
   while (result = read (from_handle, buffer, 512))
      if (result != write (to_handle, buffer, result))
      {
         fprintf (stderr, "\nError on writing target file");
         break;
      }

   close (from_handle);
   close (to_handle);
} /* copy_file ********************************************************/


void initialize_output(void)
/*
DESCRIPTION:
  initialize array of output structures
CALLED BY:
  initialize
*/
{
   int i;

   for (i = 0; i < MAX_OUTPUT_FILES; i++)
   {
      outputs[i].SAVE_OUTPUT = FALSE;
      strcpy (outputs[i].user_outputfile_name, " ");
   }
   strcpy (outputs[STABLE].model_outputfile_name, "standtbl.dls");
   strcpy (outputs[TGSTEM].model_outputfile_name, "gstemnum.dls");
   strcpy (outputs[TGBA].model_outputfile_name,   "gbasalar.dls");
   strcpy (outputs[TGVOL].model_outputfile_name,  "gvolume.dls");
   strcpy (outputs[TGDBH].model_outputfile_name,  "gdbh.dls");
   strcpy (outputs[DEBUG].model_outputfile_name,  "dbugdamg.dls");

   strcpy (model_graph_file_name,  "grafhost.dls");
```

```
   strcpy (outputs[STABLE].description, "Stand table output          ");
   strcpy (outputs[TGSTEM].description, "Stem count ASCII file       ");
   strcpy (outputs[TGBA].description,   "Basal area ASCII file       ");
   strcpy (outputs[TGVOL].description,  "Volume ASCII file           ");
   strcpy (outputs[TGDBH].description,  "Diameter ASCII file         ");
   strcpy (outputs[DEBUG].description,  "Debug file                  ");
} /* initialize_output ********************************************************/


int select_outputs(VOID *sdata, int idata)
/*
DESCRIPTION:
  edit output data
CALLED BY:
  framer menu SELECT OUTPUTS
CALLS:
  IVIEW_edit (when IPYEAR == 0);
NOTES:
  this is stored in STAND data
*/
{
  menu_type menu;
  sed_type output_sed;
  char range_string[25];   /* e.g. "(1,10)\n"  */
  boolean tables_needed[MAX_OUTPUT_FILES];
  int i;

  for (i=0; i<MAX_OUTPUT_FILES; i++)  {
     if(strcmp("t", outputs[i].output_needed)==0  ||  strcmp("T", outputs[i].output_needed)==0)
        tables_needed[i] = TRUE;
     else
        tables_needed[i] = FALSE;
  }
  EDITS_MADE = TRUE;
  file_options = file_options | GEN;
  file_options = file_options | STAND;  /* for ICAT */
  if (IGYEAR > NYEAR)
      IGYEAR = NYEAR;
  sprintf (range_string, "(1,%d)", NYEAR);

  if ((menu = menu_Open()) == NULL)
  {
     printf ("unable to open menu");
     return(0);
  }
  win_width = 70;
  win_y = 9;
  win_x = 3;
  menu_Printf(menu,"@a[0x1B]TABULAR OUTPUT:@a[0x17]\n");
  menu_Printf(menu,"  Output interval: @fd2[###]\n",
    &IPYEAR, &int_funcs,"Stand table output interval; >= 0; IPYEAR", "(0,)");
  menu_Printf(menu, " Stand Table Output? @fd[###]\n",
     &tables_needed[STABLE], &yesno_funcs, "The Space Bar toggles between YES and NO");
  menu_Printf(menu,"@a[0x1B]GRAPHICS & ASCII FILE OUTPUT:@a[0x17]\n");
  menu_Printf(menu, "  Output interval: @fd2[###]\n",
    &IGYEAR, &int_funcs,"Number greater than 0 & less than or equal to simulation length; IGYEAR",
range_string);
  menu_Printf(menu, "  Tree summary category: @fd2[###]\n",
    &ICAT, &int_funcs,"1 = overstory; 2 = understory; 3 = total; ICAT", "(1,3)");
  menu_Printf(menu, "  Stem Count File?: @fd[###]\n",
     &tables_needed[TGSTEM], &yesno_funcs, "The Space Bar toggles between YES and NO");
  menu_Printf(menu, "  Basal Area File?: @fd[###]\n",
     &tables_needed[TGBA], &yesno_funcs, "The Space Bar toggles between YES and NO");
  menu_Printf(menu, "  Volume File?: @fd[###]\n",
     &tables_needed[TGVOL], &yesno_funcs, "The Space Bar toggles between YES and NO");
  menu_Printf(menu, "  Diameter File?: @fd[###]\n",
     &tables_needed[TGDBH], &yesno_funcs, "The Space Bar toggles between YES and NO");
  menu_Printf(menu,"@a[0x1B]DEBUG FILE OUTPUT:@a[0x17]\n");
  menu_Printf(menu, "  Debug Information needed?: @fd[###]\n",
     &tables_needed[DEBUG], &yesno_funcs, "The Space Bar toggles between YES and NO");
  menu_Flush(menu);
  if ((output_sed = sed_Open(menu)) == NULL)
  {
     printf ("unable to open sed");
     return(0);
  }
  sed_SetLabel(output_sed, 31);
  sed_SetWidth(output_sed, win_width);
  sed_SetPosition(output_sed, win_y, win_x);
  sed_SetColors (output_sed, CYAN_BLUE, WHITE_BLUE, YELLOW_CYAN);
  sed_SetShadow(output_sed, 1);
  sed_SetShadowAttr(output_sed, DGRAY_BLACK);
```

```
   sed_SetMouse(output_sed, sedmou_GreedyTrack);
   sed_SetBorder(output_sed, bd_std);
   sed_SetBorderTitle(output_sed, "        OUTPUT PRODUCTION CONTROL");
   sed_Repaint(output_sed);
   sed_Go(output_sed);
   for (i=0; i<MAX_OUTPUT_FILES; i++)  {
      if (tables_needed[i] == TRUE)
         strcpy(outputs[i].output_needed, "T");
      else
         strcpy(outputs[i].output_needed, "F");
   }
   if (IPYEAR == 0)
      (void) IVIEW_edit ();
   sed_Close(output_sed);
   return (0);
} /* select_outputs ************************************************/


int IVIEW_edit (void)
/*
DESCRIPTION:
  edit IVIEW array
CALLED BY:
  select_outputs  (when IPYEAR == 0)
CALLS:
  insertion_sort, move_array_zeros
*/
{
   menu_type menu;
   sed_type iview_sed;
   int i;
   char range_string[25];        /* e.g. "(1978,2000)\n"  */
   char start_yr_string[5];      /*  "1990\n" */
   char end_yr_string[5];
   int ending_yr;

   ending_yr = ISYEAR + NYEAR - 1;
   sprintf (range_string, "(0,0)(%d,%d)", ISYEAR, ending_yr);
   sprintf (start_yr_string, "%d", ISYEAR);
   sprintf (end_yr_string, "%d", ending_yr);

   if ((menu = menu_Open()) == NULL)
   {
      printf ("unable to open menu");
      return(0);
   }
   win_width = 50;
   win_y = 15;
   win_x = 6;
   menu_Printf(menu, "Starting Year: @a[0x1F]%s@a[0x17] ",
      start_yr_string);
   menu_Printf(menu, "  Ending Year: @a[0x1F]%s@a[0x17] \n",
      end_yr_string);
   for (i = 0; i < MAX_VIEWS; i++)
   {
    menu_Printf(menu, " @fd2[#####]",
       &IVIEW[i], &int_funcs,"Specific year for stand output; IVIEW", range_string);
    if (((i+1) % 5) == 0)  /* skip line */
       menu_Printf(menu, "\n");
   }
   menu_Flush(menu);
   if ((iview_sed = sed_Open(menu)) == NULL)
      printf ("unable to open sed");
   sed_SetLabel(iview_sed, 27);
   sed_SetWidth(iview_sed, win_width);
   sed_SetPosition(iview_sed, win_y, win_x);
   sed_SetColors (iview_sed, CYAN_BLUE, WHITE_BLUE, YELLOW_CYAN);
   sed_SetShadow(iview_sed, 1);
   sed_SetShadowAttr(iview_sed, DGRAY_BLACK);
   sed_SetMouse(iview_sed, sedmou_GreedyTrack);
   sed_SetBorder(iview_sed, bd_std);
   sed_SetBorderTitle(iview_sed, " YEARS TO PRODUCE STAND TABLE ");
   sed_SetSpecial(iview_sed, inter_field_grid);
   sed_Repaint(iview_sed);
   sed_Go(iview_sed);
   sed_Close(iview_sed);

   (void) delete_array_duplicates(IVIEW, MAX_VIEWS);
   (void) insertion_sort (IVIEW, MAX_VIEWS);   /* sort array; lowest to highest*/
   (void) move_array_zeros (IVIEW, MAX_VIEWS); /* move zeros to end */
   return (0);
}  /* IVIEW_edit ***************************************************/
```

```
void delete_array_duplicates(int array[], int array_number)
/*
DESCRIPTION:
  remove duplications in array
ARGUMENTS:
  array: array to be sorted
  array_number: number of elements in array
CALLED BY:
  IVIEW_edit
*/
{
   int i, j;
   int temp;

   for(i=0; i < array_number; i++)
   {
      temp = array[i];
      if (temp != 0)                          /* ignore if 0 */
         for(j=i+1; j < array_number; j++)
            if (temp == array[j])             /* duplication? */
               array[j] = 0;                  /* set to zero */
   }
}  /* delete_array_duplicates ********************************************/


void insertion_sort (int array[], int array_number)
/*
DESCRIPTION:
  sorts integer array; lowest to highest (C TOOLBOX)
ARGUMENTS:
  array: array to be sorted
  array_number: number of elements in array
CALLED BY:
  IVIEW_edit
*/
{
   int i, j;
   int temp;

   for(i=1; i < array_number; i++)
   {                                       /* insert the i-th element */
        temp = array[i];                   /* into the sorted array */
        j = i - 1;                              /* find where new element goes */
        while( (j >= 0) && (temp < array[j]) )
        {
            array[j+1] = array[j];         /* not here - move this one up */
            j = j - 1;
        }
        array[j+1] = temp;
    }
}  /* insertion_sort *****************************************************/


void move_array_zeros (int array[], int array_number)
/*
DESCRIPTION:
  moves zeros in array to the end of the array (after sorting)
ARGUMENTS:
  array: array to be sorted
  array_number: number of elements in array
CALLED BY:
  IVIEW_edit
CALLS:
  exchang_values
*/
{
   int i,
   zero_num;  /* number of zero elements; they will be at beginning */

   zero_num = 0;
   for (i = 0; i < array_number; i++)
     if (array[i] < 0.01)
        zero_num = i+1;
   if ((zero_num < array_number) && (zero_num > 0))
      for (i = 0; i < (array_number - zero_num); i++)
        (void) exchange_values (&array[i], &array[i+zero_num]);
} /* move_array_zeros ***************************************************/


void exchange_values (int *ptr_A, int *ptr_B)
/*
```

```
DESCRIPTION:
  moves zeros in array to the end of the array (after sorting)
ARGUMENTS:
  ptr_A, ptr_B: values to switch
CALLED BY:
  move_array_zeros
*/
{
   int temp;

   temp = *ptr_A;
   *ptr_A = *ptr_B;
   *ptr_B = temp;
} /* exchange_values ********************************************************/


int choose_view_or_print (VOID *sdata, int idata)
/*
DESCRIPTION:
  choose an output file, then view or print it
CALLED BY:
  framer menu VIEW OUTPUTS or PRINT OUTPUTS (based on idata)
CALLS:
  view_output, print_output
*/
{
   int i;
   int returned_from_sed;  /* sed_Go return value */
   int file_index;         /* index of output files */
   boolean OUTPUT_EXISTS;  /* output exists on disk */
   menu_type menu;
   sed_type choose_sed;

   /* do any output files exist? */
   for (i = 0; i < MAX_OUTPUT_FILES; i++)
      if ((access(outputs[i].model_outputfile_name, 0)) == 0)
         OUTPUT_EXISTS = TRUE;

   if (TRUE == OUTPUT_EXISTS)
   {
     if ((menu = menu_Open()) == NULL)
     {
        printf ("unable to open menu");
        return(0);
     }
     win_width = 60;
     win_y = 13;
     win_x = 6;
     if ((access(outputs[STABLE].model_outputfile_name, 0)) == 0)
        menu_Printf(menu,"@fd2[Stand table output   ]\n", NULL, &menu_funcs, NULL, "1");
     if ((access(outputs[TGSTEM].model_outputfile_name, 0)) == 0)
        menu_Printf(menu,"@fd2[Stem count ASCII file]\n", NULL, &menu_funcs, NULL, "2");
     if ((access(outputs[TGBA].model_outputfile_name, 0)) == 0)
        menu_Printf(menu,"@fd2[Basal area ASCII file]\n", NULL, &menu_funcs, NULL, "3");
     if ((access(outputs[TGVOL].model_outputfile_name, 0)) == 0)
        menu_Printf(menu,"@fd2[Volume ASCII file    ]\n", NULL, &menu_funcs, NULL, "4");
     if ((access(outputs[TGDBH].model_outputfile_name, 0)) == 0)
        menu_Printf(menu,"@fd2[Diameter ASCII file  ]\n", NULL, &menu_funcs, NULL, "5");
     if ((access(outputs[DEBUG].model_outputfile_name, 0)) == 0)
        menu_Printf(menu,"@fd2[Debug file           ]",   NULL, &menu_funcs, NULL, "6");
     menu_Flush(menu);
     if ((choose_sed = sed_Open(menu)) == NULL)
     {
        printf ("unable to open sed");
        return(0);
     }
     if (idata == VIEW)
        sed_SetLabel (choose_sed, 46);
     else /* == PRINT */
        sed_SetLabel (choose_sed, 60);
     sed_SetWidth(choose_sed, win_width);
     sed_SetPosition(choose_sed, win_y, win_x);
     sed_SetColors (choose_sed, CYAN_BLUE, WHITE_BLUE, YELLOW_CYAN);
     sed_SetShadow(choose_sed, 1);
     sed_SetShadowAttr(choose_sed, DGRAY_BLACK);
     sed_SetMouse(choose_sed, sedmou_GreedyTrack);
     sed_SetBorder(choose_sed, bd_title);
     if (idata == VIEW)
        sed_SetBorderTitle(choose_sed, "Use ARROW keys to highlight your choice, then press ENTER");
     else /* == PRINT */
        sed_SetBorderTitle(choose_sed, "Insure printer is ready, highlight choice, then press ENTER");
     sed_Repaint(choose_sed);
```

```
      returned_from_sed = sed_Go(choose_sed);
      while (returned_from_sed > 0)
      {
         file_index = returned_from_sed -1;
         if (idata == VIEW)
            (void) view_output(file_index);
         else /* == PRINT */
            (void) print_output(file_index);
         sed_Repaint(choose_sed);
         returned_from_sed = sed_Go(choose_sed);
      }
      sed_Close(choose_sed);
    }
    else
    {
      beep();
      prompt(" No output files found.\n (Select options, then Run model)");
    }
    return (0);
} /* choose_view_or_print ***************************************************/


void view_output (int file_index)
/*
DESCRIPTION:
  view output file
ARGUMENTS:
  file_index indicates output file
CALLED BY:
  choose_view_or_print
*/
{
  int i;
  static char text[OUTPUT_BYTES];
  char temp_description[31];
  int char_length;

  /* temp_description for shorter string for pop_View */
  i = 0;
  while ((outputs[file_index].description[i]   != ' ') ||
         (outputs[file_index].description[i+1] != ' '))
           i++;
  char_length = i;
  strncpy (temp_description, outputs[file_index].description, char_length);
  temp_description[char_length] = '\0';

  outputs[file_index].outputfile = fopen (outputs[file_index].model_outputfile_name, "r");
  if (outputs[file_index].outputfile == (FILE *) NULL)
     prompt ("Cannot open selected file");
  else
  {
     yellow_message ("One Moment Please");
     /* insert NULL at end of text to prepare for pop_View */
     text[fread (text, 1, OUTPUT_BYTES, outputs[file_index].outputfile)] = '\0';
     pop_View3(temp_description,
               text, 0, 0, disp_GetHeight()-4, disp_GetWidth()-2, YELLOW_BLUE, 52+file_index, bd_mouse2);
     yellow_message (NULL);
     fclose(outputs[file_index].outputfile);
  }
} /* view_output ************************************************/


int print_output (int file_index)
/*
DESCRIPTION:
  print output file
ARGUMENTS:
  file_index indicates output file
CALLED BY:
  choose_view_or_print
CALLS:
  printer_test, copy_file, verify_choice
NOTES:
  has 2 exits
*/
{
  unsigned bit_status; /* bits indicate printer result */
  int printer_status;

  if ((access(outputs[file_index].model_outputfile_name, 0)) != 0)
     prompt ("Cannot find selected file");
  else
```

```
   {
      yellow_message (" Print request made...Please Standby ");
      printer_status = printer_test();

      if (SUCCESS != printer_status)
      {
         yellow_message (NULL);
         CHOICE = FALSE;
         if (printer_status == CANNOT_INIT)
            verify_choice ("    PROCEED with Print Job?",
                           "WARNING: Be sure printer is ready.");
         else
            verify_choice ("PROCEED with Print Job?",
                           "WARNING: Printer not responding as expected.");
         if (CHOICE == FALSE)
            return(0);
         /* else give msg and proceed with copy */
         yellow_message (" Proceeding with print request...Please Standby ");
      }
      copy_file (outputs[file_index].model_outputfile_name, "prn");

      bit_status = callprint(PRINT_CHAR, 12, PORT_LPT1);        /* form feed */
      if ((bit_status ^ SELECTED) != 0 )
         prompt ("Form feed unsuccessful");
      yellow_message (NULL);
      prompt ("             WARNING:\nWAIT until printing is done before\nprinting another file.");
   }
   return(0);
} /* print_output ************************************************/


int printer_test(void)
/*
DESCRIPTION:
  insures that parallel printer is turned on and ready
CALLED BY:
  print_output
RETURNS:
  0 if successful
NOTES:
  uses BIOS INT 17; hard_Pause so printer can respond; 3 exits
*/
{
  unsigned status; /* bits indicate result */

  /* initialize printer & port */
  status = callprint(INIT_PTR, 0, PORT_LPT1);
  if ((status ^ INIT_OK) != 0 )
     return(CANNOT_INIT);
  hard_Pause(300);

  /* get printer status */
  status = callprint(GET_STATUS, 0, PORT_LPT1);
  if ((status ^ NOT_BUSY_and_SELECTED) != 0 )
     return(NOT_READY);

  return(0);
} /* printer_test ************************************************************/


int output_to_save (VOID *sdata, int idata)
/*
DESCRIPTION:
  choose output files to save
CALLED BY:
  framer menu SAVE OUTPUT FILES, delete_output_after_save
CALLS:
  create_outputfile_name, copy_file
*/
{
   int i;
   int blank_entered_count[6];
   boolean OUTPUT_EXISTS;        /* output files exists on disk */
   char file_msg_string[80];
   menu_type menu;
   sed_type output_sed;

   /* do any output files exist?*/
   for (i = 0; i < MAX_OUTPUT_FILES; i++)
      if ((access(outputs[i].model_outputfile_name, 0)) == 0)
      {
         blank_entered_count[i] = 0;
```

```
      OUTPUT_EXISTS = TRUE;
   }
if (TRUE == OUTPUT_EXISTS)
{
  if ((menu = menu_Open()) == NULL)
  {
     printf ("unable to open menu");
     return(0);
  }
  win_width = 42;
  win_y = 7;
  win_x = 6;
  for (i = 0; i < MAX_OUTPUT_FILES; i++)
  {
     /* does file exist? */
     if ((access(outputs[i].model_outputfile_name, 0)) == 0)
        menu_Printf(menu, "%s @fd[###]\n", outputs[i].description,
           &outputs[i].SAVE_OUTPUT, &yesno_funcs, "The Space Bar toggles between YES and NO");
     else
        outputs[i].SAVE_OUTPUT = FALSE;
  }
  menu_Flush(menu);
  if ((output_sed = sed_Open(menu)) == NULL)
  {
     printf ("unable to open sed");
     return(0);
  }
  sed_SetLabel (output_sed, 47);
  sed_SetWidth(output_sed, win_width);
  sed_SetPosition(output_sed, win_y, win_x);
  sed_SetColors (output_sed, CYAN_BLUE, WHITE_BLUE, YELLOW_CYAN);
  sed_SetShadow(output_sed, 1);
  sed_SetShadowAttr(output_sed, DGRAY_BLACK);
  sed_SetMouse(output_sed, sedmou_GreedyTrack);
  sed_SetBorder(output_sed, bd_std);
  sed_SetBorderTitle(output_sed, "  Choose Files to Save");
  sed_Repaint(output_sed);
  sed_Go(output_sed);
  for (i = 0; i < MAX_OUTPUT_FILES; i++)
  {
     if (outputs[i].SAVE_OUTPUT == TRUE)
     {
        while ((strcmp(" ", outputs[i].user_outputfile_name)==0) &&
               (blank_entered_count[i] < 1))
        {
           if (blank_entered_count[i] > 0)
              beep();
           create_outputfile_name (i);
           blank_entered_count[i]++;
        }
        outputs[i].SAVE_OUTPUT = FALSE;

        if (strcmp(" ", outputs[i].user_outputfile_name)==0)
        {
           beep();
           sprintf(file_msg_string,
              "          %s\n          WARNING: File Not Saved", outputs[i].description);
           prompt (file_msg_string);
        }
        /* does file exist? */
        else if ((access(outputs[i].user_outputfile_name, 0)) == 0)
        {
           CHOICE = FALSE;
           verify_choice ( "OVERWRITE existing File?",
                           "WARNING: File exists on disk");
           if (CHOICE == TRUE)
           {
              /* overwrite file */
              sprintf (file_msg_string, "Saving output file: %s",
                                        outputs[i].user_outputfile_name);
              yellow_message (file_msg_string);
              /* copy file to new name */
              copy_file (outputs[i].model_outputfile_name,
                         outputs[i].user_outputfile_name);
              yellow_message (NULL);
           }
           else /* don't save file, give WARNING, initialize file name */
           {
              beep();
              sprintf(file_msg_string,
                      "          %s\n          WARNING: File Not Saved",
                     outputs[i].description);
```

```
                prompt (file_msg_string);
                strcpy(outputs[i].user_outputfile_name, " ");
             }
          }
          else /* file does not exist, go ahead and save */
          {
             sprintf (file_msg_string, "Saving output file: %s", outputs[i].user_outputfile_name);
             yellow_message (file_msg_string);
             /* copy file to new name */
             copy_file (outputs[i].model_outputfile_name, outputs[i].user_outputfile_name);
             yellow_message (NULL);
          }
       }
     }
     sed_Close(output_sed);
   }
   else
   {
     beep();
     prompt(" No output files found.\n (Select options, then Run model)");
   }
   return (0);
} /* output_to_save ******************************************************/


void create_outputfile_name(int file_index)
/*
DESCRIPTION:
  edits output file name
ARGUMENTS:
  file_index is the index of the outputs array
SAMPLE CALL:
  output_file_edit ( file_index );
CALLED BY:
  output_to_save
CALLS:
  check_drive_letter, remove_invalid_chars
*/
{
   menu_type menu;
   sed_type file_sed;
   int file_length_error=FALSE;

   do{
      if (file_length_error) {
         strcpy (outputs[file_index].user_outputfile_name, " ");
         file_length_error = FALSE;
      }
      if ((menu = menu_Open()) == NULL)
         printf ("unable to open menu");
      win_width = 68;
      win_y = 20;
      win_x = 3;
      menu_Printf(menu, "Disk File Name: @fd[############]\n",
         outputs[file_index].user_outputfile_name, &string_funcs,
         "Left justified File Name   (CAUTION: Do not overwrite existing file)");
      menu_Flush(menu);
      if ((file_sed = sed_Open(menu)) == NULL)
         printf ("unable to open sed");
      sed_SetLabel(file_sed, 48);
      sed_SetPosition(file_sed, win_y, win_x);
      sed_SetColors (file_sed, WHITE_BLUE, WHITE_BLUE, YELLOW_CYAN);
      sed_SetShadow(file_sed, 1);
      sed_SetShadowAttr(file_sed, DGRAY_BLACK);
      sed_SetWidth(file_sed, win_width);
      sed_SetMouse(file_sed, sedmou_GreedyTrack);
      sed_SetBorder(file_sed, bd_std);
      sed_SetBorderTitle(file_sed, outputs[file_index].description);
      sed_Repaint(file_sed);
      sed_Go(file_sed);
      sed_Close(file_sed);
      if (strcmp(" ", outputs[file_index].user_outputfile_name)!=0) { /*name entered*/
         check_drive_letter   (outputs[file_index].user_outputfile_name);
         remove_invalid_chars (outputs[file_index].user_outputfile_name);
         if (file_length_error = check_file_length(outputs[file_index].user_outputfile_name)) {
            beep();
            prompt(" ERROR: File name entered incorrectly.\n Please try again.");
         }
      }
   }while(file_length_error);
} /* create_outputfile_name **********************************************/
```

```
/*****************************************/
int check_file_length(char file_name[13])
/*****************************************
DESCRIPTION: don't let initial part of file exceed 8 characters
*/
{
char dot = '.';
int i, len, dot_position=0;

   /* no problem if 8 or less characters */
   len = strlen(file_name);
   if (len < 9)
      return(0);

   /* if dot past 8th char, return ERROR */
   for (i = 0; i < len; i++)  {
      if (file_name[i] == dot)
         dot_position = i+1;
      if (dot_position > 9)
         return(ERROR);
   }

   /* if dot not found, return error */
   if (dot_position == 0)
      return(ERROR);
   return(0);
}


void check_drive_letter (char file_name[9])
/*
DESCRIPTION:
  checks for A: or B: or C:
ARGUMENTS:
  DOS file name
RETURNS:
  file_name with d_ instead of d:
CALLED BY:
  create_outputfile_name
SAMPLE CALL:
  check_drive_letter ( "a:file1" );
*/
{
   char valid_chars[NUM_VALID_CHARS] =
      "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789_-.:\0";
   int do_nothing; /* does nothing, use for if statement */

   /* Allow valid drive of "A:" or "B:" or "C:" */
   if (file_name[1] == ':')
   {
      if ((file_name[0] == 'a') ||
          (file_name[0] == 'A') ||
          (file_name[0] == 'b') ||
          (file_name[0] == 'B') ||
          (file_name[0] == 'c') ||
          (file_name[0] == 'C'))
             do_nothing = 0;
      else
      {
          prompt(" Invalid drive specification");
          file_name[1] = '_';
      }
   }
} /* check_drive_letter ************************************************/


void delete_output_after_save(void)
/*
DESCRIPTION:
  check output file status before deleting them
CALLED BY:
  exit_routine, run_model, get_file, choose_inp_file, reload_default_file
CALLS:
  output_to_save
*/
{
   int i;
   int result;
   boolean NOT_SAVED = FALSE;

   /* does file exist AND user has not saved it? */
   for (i = 0; i < MAX_OUTPUT_FILES; i++)
```

```
      if (((access(outputs[i].model_outputfile_name, 0))==0) &&
          (outputs[i].SAVE_OUTPUT == TRUE))
              NOT_SAVED = TRUE;

   /* give user a chance to save output files */
   if (TRUE == NOT_SAVED)
   {
      CHOICE = TRUE;
      verify_choice ("SAVE Existing Output?",
                     "WARNING: Temporary Output Files exist on disk");
      if (CHOICE == TRUE)
      (void) output_to_save(NULL, 0);
   }

   /* remove files if they exist */
   for (i = 0; i < MAX_OUTPUT_FILES; i++)
      if ((access(outputs[i].model_outputfile_name, 0))==0)
      {
         result = remove(outputs[i].model_outputfile_name);
         if (result == -1)
            printf ("%s file could not be opened.\n", outputs[i].model_outputfile_name );
      }
   if ((access(model_graph_file_name, 0))==0)
   {
      result = remove(model_graph_file_name);
      if (result == -1)
         printf ("%s file could not be opened.\n", model_graph_file_name);
   }
} /*  delete_output_after_save ************************************************/
```

```
/***************** DRAW VARIABLES HEADER FILE **********/
#define SOLID           0xFFFF
/* COLORS */
#define BLACK           0
#define BLUE            1
#define GREEN           2
#define CYAN            3
#define RED             4
#define MAGENTA         5
#define LIGHT_GRAY      7
#define COLORS_USED     13

#define SINGLE          1
#define DOUBLE          2
#define NFONTS          6
#define COURIER         0
#define ROMAN           5


/*************************************************************************/
/* CSCAPE window variables */
extern int win_width, /* window width */
    win_x,            /* window position (x coord) of upper left hand corner */
    win_y,            /* window position (y coord) of upper left hand corner */
    win_height;       /* window height */

/* seds to close to save memory for graphics */
extern sed_type
    file_heading_sed,  /* used for displaying file */
    heading_sed,       /* upper heading */
    screen_help_sed;   /* displays help line at screen bottom */
/* for file_heading_sed */
extern char forestfile_name[9],
       long_forestfile_name[13];
extern int mouse_works;
/*************************************************************************/

/* model generated GENERAL data file variables */
int graph_NHOSTS;

/* individual HOST SPECIES data file variables */
char graph_TNAME[MAX_HOSTS][3];

/*************************************************************************/
/* DAMAGE MODEL OUTPUT variables */


/* output files data structure */
struct output_file_str {
   char output_needed[2];     /* FORTRAN LOGICAL var; treated as 1 char string */
   boolean SAVE_OUTPUT;       /* TRUE if user wants to save output file */
   char description[31];      /* description of output that is needed */
   FILE *outputfile;          /* file created by damage model */
   char model_outputfile_name[13]; /* name of file created by damage model */
   char user_outputfile_name[13];  /* renamed file created by damage model */
};

/* array of structures */
extern struct output_file_str  outputs[MAX_OUTPUT_FILES];


/* graphic output files data structure */
struct graphic_output_str {
   char graph_heading[40];         /* heading of graph */
   char label[12];                 /* Y label for graphics */
};

struct graphic_output_str    graphic_outputs[MAX_OUTPUT_FILES];


extern char  model_graph_file_name[13]; /* file generated by model; contains NHOSTS */
extern FILE *model_graph_file;
/*************************************************************************/
/* GRAPHICS variables */

PRIVATE VOID   *textdisp, *grafdisp;   /* to switch between graphics & text */
int ErrorCode;                         /* Reports any graphics errors       */
extern int screen_color,
   graphics_capable,
   command_line_screen_color;

int year_count;                  /* number of separate years of output */
```

```
int years[MAX_SIM_YRS];            /* array of years for X axis */
float  Y[MAX_HOSTS][MAX_SIM_YRS];  /* 2D array of amounts for Y axis */
extern int units;

/*************************************************************************/
/* GRAPHICS VIEWPORT/WINDOWS *********************************************/

/* viewport structure */
typedef struct v_str {
   int screenXmax;      /* X axis screen size (pixels) - resolution */
   int screenYmax;      /* Y axis screen size (pixels) - resolution */
   int textW;         /* in pixels */
   int textH;        /* in pixels */
   double aspect_ratio; /* screen aspect ratio */
   int max_colors;      /* screen available colors */
   int colors[COLORS_USED];
} VIEW_PORT;
VIEW_PORT v;

float Wxl, Wxr, Wyb, Wyt,       /* world coordinates */
      Vxl, Vxr, Vyb, Vyt,       /* viewport coordinates */
      WVxm, WVxa, WVym, WVya;   /* matrix multiplication and addition */
float max_X, min_X,             /* maximum/minimum values of world coordinates */
      max_Y, min_Y;
```

```
/********************* GRAPH OUTPUT ************************/
/* Microsoft C 6.0 includes */
#include <stdio.h>
#include <stdlib.h>     /* itoa */
#include <string.h>     /* contains declarations for the string handling fncs */
#include <io.h>         /* for access */
#include <graph.h>
#include <dos.h>
/* CSCAPE 3.2 includes */
#include <cscape.h>
#include <popdecl.h>
/* define file */
#include "defines.h"
#include "draw.h"

/* FUNCTION PROTOTYPES <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<*/
void read_XYcoord (FILE *fptr);
void read_graph_NHOSTS (FILE *fptr);
PRIVATE int init_graphics(void);
PRIVATE int set_video_mode(void);
PRIVATE void map_colors (int color_num);
PRIVATE void system_inquire (void);
PRIVATE void close_graphics(void);
PRIVATE void init_graphic_outputs(void);
PRIVATE void init_window_viewport();
PRIVATE void init_graph_parameters(void);
PRIVATE void changetextstyle (int height, int width, int fontnum);
PRIVATE void set_window_viewport ();
PRIVATE void set_window (float x1, float y1, float x2, float y2);
PRIVATE void my_set_viewport (float x1, float y1, float x2, float y2);
PRIVATE float X_to_screen (float x);
PRIVATE float Y_to_screen (float y);
PRIVATE void line (float X1, float Y1, float X2, float Y2, int thickness);
PRIVATE void XYGraph(int file_index);

/* routines using CSCAPE function calls */
int choose_output_for_draw (VOID *sdata, int idata);
void control_output_draw (int file_index);
void screen_help(void);


int choose_output_for_draw (VOID *sdata, int idata)
/*
DESCRIPTION:
  choose an output file, then draw it
CALLED BY:
  framer menu GRAPH OUTPUTS
CALLS:
  control_output_draw
*/
{
   int i;
   int returned_from_sed;  /* sed_Go return value */
   int file_index;         /* index of output files */
   boolean OUTPUT_EXISTS;  /* output exists on disk */
   menu_type menu;
   sed_type draw_sed;

   /* ASCII graphics files exist? */
   for (i = TGSTEM; i <= TGDBH; i++)
      if ((access(outputs[i].model_outputfile_name, 0)) == 0)
         OUTPUT_EXISTS = TRUE;

   if (TRUE == OUTPUT_EXISTS)
   {
     if ((menu = menu_Open()) == NULL)
     {
        printf ("unable to open menu");
        return(0);
     }
     win_width = 60;
     win_y = 13;
     win_x = 6;
     if ((access(outputs[TGSTEM].model_outputfile_name, 0)) == 0)
        menu_Printf(menu,"@fd2[Stem count ASCII file]\n", NULL, &menu_funcs, NULL, "2");
     if ((access(outputs[TGBA].model_outputfile_name, 0)) == 0)
        menu_Printf(menu,"@fd2[Basal area ASCII file]\n", NULL, &menu_funcs, NULL, "3");
     if ((access(outputs[TGVOL].model_outputfile_name, 0)) == 0)
        menu_Printf(menu,"@fd2[Volume ASCII file    ]\n", NULL, &menu_funcs, NULL, "4");
     if ((access(outputs[TGDBH].model_outputfile_name, 0)) == 0)
        menu_Printf(menu,"@fd2[Diameter ASCII file  ]\n", NULL, &menu_funcs, NULL, "5");
     menu_Flush(menu);
```

```
      if ((draw_sed = sed_Open(menu)) == NULL)
      {
         printf ("unable to open sed");
         return(0);
      }
      sed_SetLabel (draw_sed, 58);
      sed_SetWidth(draw_sed, win_width);
      sed_SetPosition(draw_sed, win_y, win_x);
      sed_SetColors (draw_sed, CYAN_BLUE, WHITE_BLUE, YELLOW_CYAN);
      sed_SetShadow(draw_sed, 1);
      sed_SetShadowAttr(draw_sed, DGRAY_BLACK);
      sed_SetMouse(draw_sed, sedmou_GreedyTrack);
      sed_SetBorder(draw_sed, bd_title);
      sed_SetBorderTitle(draw_sed, "Use ARROW keys to highlight your choice, then press ENTER key");
      sed_Repaint(draw_sed);
      returned_from_sed = sed_Go(draw_sed);
      while (returned_from_sed > 0)  {
         file_index = returned_from_sed -1;
         yellow_message ("Creating Graphics\nOne Moment Please...");
         hard_Pause (100);
         yellow_message (NULL);
         (void) control_output_draw (file_index);
         sed_Repaint(draw_sed);
         returned_from_sed = sed_Go(draw_sed);
      }
      sed_Close(draw_sed);
   }
   else
   {
     beep();
     prompt(" No graphic output files found.\n (Select options, then Run model)");
   }
   return (0);
} /* choose_output_for_draw ***************************************************/


void control_output_draw (int file_index)
/*
DESCRIPTION:
  manage the graphing of output
ARGUMENT:
  file_index refers to particular graphic file
CALLED BY:
  choose_output_for_draw
CALLS:
  init_graphics, init_graph_parameters, read_XYcoord, read_graph_NHOSTS,
  XYGraph, init_window_viewport, close_graphics
*/
{
   /* are 2 files present? */
   outputs[file_index].outputfile = fopen (outputs[file_index].model_outputfile_name, "r");
   model_graph_file = fopen (model_graph_file_name, "r");
   if ((outputs[file_index].outputfile == (FILE *)NULL) || (model_graph_file == (FILE *)NULL))
   {
      if (model_graph_file == (FILE *) NULL)
         prompt("Cannot open host file generated by model.");
      else
         prompt ("Cannot open selected file.");
   }
   else
   {
      (void) init_graph_parameters();
      (void) read_graph_NHOSTS (model_graph_file);
      (void) read_XYcoord (outputs[file_index].outputfile);

      if (year_count > 1)       /* need at least 2 points on graph */
      {
         (void) init_graphic_outputs();
         (void) init_window_viewport();

         /* close sed windows to save memory for graphics */
         sed_Close (file_heading_sed);
         sed_Close (heading_sed);
         sed_Close (screen_help_sed);

         disp_HideMouse();
         if (SUCCESS == init_graphics())  {
            (void) XYGraph(file_index);
            /* if HELP, redraw after */
            while (FN1 == kb_Read())  {
               help_Show (59, file_index);
               (void) XYGraph(file_index);
```

```
            }
            (void) close_graphics();
            if (graphics_capable)
               disp_Repaint();
         }
         else
             prompt ("Unable to show graphics");
         disp_ShowMouse();

         /* reopen sed windows after running model */
         heading_sed = heading_text
            ("STAND DAMAGE MODEL: part of the Gypsy Moth Life System Model   Version 1.1");
         if (strcmp(" ", forestfile_name)!=0)    /* user has entered name */
            file_heading_sed = file_text (long_forestfile_name);
         else
            file_heading_sed = file_text ("default");
         screen_help();
      }
      else
         prompt ("Need at least 2 years to draw graph:\nRerun model with more years");
   }
} /* control_output_draw ***************************************************/


int init_graphics(void)
/*
DESCRIPTION:
  Initializes graphics system
CALLED BY:
  control_output_draw
RETURNS:
  -1 when error occurs
CALLS:
  system_inquire, close_graphics, set_video_mode, map_colors
*/
{
  int scancode;
  static int INITIALIZE_ONCE = 0;  /* static so it is zero only once */

  if (!graphics_capable)
     textdisp = disp_GetCurrent();  /* Save pointer to text interface */

  if (0 == INITIALIZE_ONCE) {   /* initialize only once */
      if (!graphics_capable) {
         /* init Cscape device interface: def_ModeGraphics selects best graphics mode*/
         if (!disp_Init(def_ModeGraphics, FNULL)) {
              printf("\n No graphics hardware found or there is insufficient memory");
            scancode = getch();
            return(-1); /* exit */
         }
      }

      /* read header info from .FON files in directory (p.528 MS C run-time library)*/
      if (_registerfonts("COURB.FON") <= 0) {
         prompt ("ERROR: Cannot find COURB.FON files; can't register font");
         (void) close_graphics();
         return(-1); /* exit */
      }
      if (_registerfonts("ROMAN.FON") <= 0) {
         prompt ("ERROR: Cannot find ROMAN.FON files; can't register font");
         (void) close_graphics();
         return(-1); /* exit */
      }

      if ((set_video_mode()) != SUCCESS)  {
         prompt ("ERROR: Cannot set graphics mode with this monitor");
         (void) close_graphics();
         return(-1); /* exit */
      }

      /* map colors for 2 color video modes */
      if ((screen_color == VGA_MONOCHROME) ||
          (disp_GetColors() == 2L) ||
          (command_line_screen_color == VGA_MONOCHROME))
            map_colors(2);
      else
            map_colors(COLORS_USED);

      INITIALIZE_ONCE++;
  }
  else if (!graphics_capable)
     /* make graphics current again */
```

```
      disp_SetCurrent(grafdisp);

   (void) system_inquire ();

   ErrorCode = _grstatus();
   if (ErrorCode < _GROK)
   {
     prompt(" SORRY: Graphics System Error");
     (void) close_graphics();
     return(-1);  /* exit */
   }
   return(0);
} /* init_graphics ************************************************************/


int set_video_mode(void)
/*
DESCRIPTION:
  Initializes video mode
CALLED BY:
  init_graphics
RETURNS:
  -1 when unsupported hardware configuration
NOTES: from CSCAPE demodraw.c
*/
{
   int mode;

   switch(pc_GetMode())
   {
      case 0x13: mode = _MRES256COLOR;
                 break;
      case 0x12: mode = _VRES16COLOR;
                 break;
      case 0x11: mode = _VRES2COLOR;
                 break;
      case 0x10: mode = _ERESCOLOR;
                 break;
      case 0x0f: mode = _ERESNOCOLOR;
                 break;
      case 0x0e: mode = _HRES16COLOR;
                 break;
      case 0x0d: mode = _MRES16COLOR;
                 break;
      case 0x06: mode = _HRESBW;
                 break;
      case 0x05:
      case 0x04: mode = _MRESNOCOLOR;
                 break;
      case 0x10A:
      case 0x10B:/* Hercules graphics card is not supported by Microsoft 5.1.
                 _HERCMONO is in their graphic.h but doesn't work with _setvideomode*/
                 mode = _HERCMONO;
                 break;

      case 0x140: /*Compaq Plasma 40 not supported by Microsoft, fall through ... */

      default: return(-1);
   }

   /* initialize Microsoft video mode */
   if ((_setvideomode(mode)) == 0)
   {

      return(-1);        /* unsupported hardware configuration */
   }
   return(0);
} /* set_video_mode ************************************************************/


void close_graphics(void)
/*
DESCRIPTION:
  removes graphics system, go back to text
CALLED BY:
  control_output_draw, init_graphics
*/
{
   if (!graphics_capable) {
      /* shut down graphics mode (remember it); back to text mode */
      grafdisp = disp_GetCurrent();
      disp_SetCurrent(textdisp);
```

```
   }
} /* close_graphics ************************************************************/


void map_colors (int color_num)
/*
DESCRIPTION:
  set colors for graphics
CALLED BY:
  init_graphics
*/
{
  int i;

  if (2 == color_num)
  {
     v.colors[0] = BLACK;
     for (i = 1; i < COLORS_USED; ++i)
        v.colors[i] = LIGHT_GRAY;
  }
  else
  {
     v.colors[0] = LIGHT_GRAY;    /* background */
     v.colors[1] = RED;  v.colors[2] = CYAN;  v.colors[3] = GREEN;
     v.colors[4] = BLACK;v.colors[5] = BLUE;  v.colors[6] = MAGENTA;
     v.colors[7] = 9;    v.colors[8] = 10;    v.colors[9] = 11;
     v.colors[10]= 12;   v.colors[11]= 13;    v.colors[12]= 14;
  }
} /* map_colors ****************************************************************/


void system_inquire (void)
/*
DESCRIPTION:
  what can monitor do?
CALLED BY:
  control_output_draw
*/
{
  struct videoconfig vc;

  _getvideoconfig (&vc);
  v.max_colors = vc.numcolors;                /* Read maximum number of colors*/
  v.screenXmax = vc.numxpixels;
  v.screenYmax = vc.numypixels;             /* Read size of screen                */
} /* system_inquire ************************************************************/


void init_graphic_outputs(void)
/*
DESCRIPTION:
  initialize array of output structures
CALLED BY:
  control_output_draw
*/
{
   if (units == SI)  {
      strcpy (graphic_outputs[TGSTEM].graph_heading, "Stem Count per ha versus Years");
      strcpy (graphic_outputs[TGBA].graph_heading,   "Basal Area (sq m/ha) vs. Years");
      strcpy (graphic_outputs[TGVOL].graph_heading,  "Volume (cu m/ha) versus Years");
      strcpy (graphic_outputs[TGDBH].graph_heading,  "   Diameter (cm) versus Years");
   }
   else {  /* english */
      strcpy (graphic_outputs[TGSTEM].graph_heading, "Stem Count per acre versus Years");
      strcpy (graphic_outputs[TGBA].graph_heading,   "Basal Area (sq ft/acre) vs. Years");
      strcpy (graphic_outputs[TGVOL].graph_heading,  "Volume (cu ft/acre) versus Years");
      strcpy (graphic_outputs[TGDBH].graph_heading,  "   Diameter (in.) versus Years");
   }
   strcpy (graphic_outputs[TGSTEM].label, "Stem Count");
   strcpy (graphic_outputs[TGBA].label,   "Basal Area");
   strcpy (graphic_outputs[TGVOL].label,  "Volume");
   strcpy (graphic_outputs[TGDBH].label,  "Diameter");
} /* init_graphic_outputs ******************************************/


void init_graph_parameters(void)
/*
DESCRIPTION:
  initialize variables
CALLED BY:
  control_output_draw
*/
```

```
{
   int i, j;

   year_count = 0;
   for ( i = 0; i < MAX_SIM_YRS; ++i )
   {
      years[i] = 0;
      for (j = 0; j < graph_NHOSTS; j++)
         Y[j][i] = 0;
   }
} /* init_graph_parameters ***********************************************/


/* VIEWPORT/WINDOWS TRANSFORMATIONS <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<*/
void set_window_viewport ()
/*
DESCRIPTION:
  compute transformation matrix with latest window and viewport parameters
CALLED BY:
  set_window, my_set_viewport
*/
{
   WVxm = (Vxr - Vxl)/(Wxr - Wxl);
   WVxa = Vxl - (Wxl * WVxm);
   WVym = (Vyt - Vyb)/(Wyt - Wyb);
   WVya = Vyb - (Wyb * WVym);
}  /* set_window_viewport ************************************************/


void set_window (float x1, float y1, float x2, float y2)
/*
DESCRIPTION:
   set world coordinate system (picture defined)
CALLED BY:
   init_window_viewport
CALLS:
   set_window_viewport
*/
{
   Wxl = x1; Wxr = x2; Wyb = y1; Wyt = y2;
   set_window_viewport ();
}  /* set_window *********************************************************/


void my_set_viewport (float x1, float y1, float x2, float y2)
/*
DESCRIPTION:
   set screen coordinates (my as opposed to TC viewport fnc)
CALLED BY:
   init_window_viewport
CALLS:
   set_window_viewport
NOTES:
   changed Vyb to Vyt and vice versa
*/
{
   Vxl = x1; Vxr = x2; Vyt = y1; Vyb = y2;
   set_window_viewport ();
}  /* my_set_viewport ****************************************************/


float X_to_screen (float x)
/*
DESCRIPTION:
  convert X World coordinates to Screen
CALLED BY:
  XYGraph
*/
{
   float xScrn;
   xScrn = WVxm * x + WVxa;
   return (xScrn);
} /* X_to_screen ********************************************************/


float Y_to_screen (float y)
/*
DESCRIPTION:
  convert Y World coordinates to Screen
CALLED BY:
  XYGraph
*/
```

```
{
   float yScrn;
   yScrn = WVym * y + WVya;
   return (yScrn);
} /* Y_to_screen ***************************************************/


void init_window_viewport ()
/*
DESCRIPTION:
  find minimums and maximums and use these to define world coordinates
CALLED BY:
  control_output_draw
CALLS:
  set_window, my_set_viewport
*/
{
   int i, j;

   /* find maximum and minimum data values */
   max_X = years[0];
   max_Y = Y[0][0];
   min_X = years[0];
   min_Y = 0;  /* Y axis must always begin with zero */

   for ( i = 0; i < year_count; ++i )
   {
      if (years[i] > max_X)
         max_X = years[i];
      if (years[i] < min_X)
         min_X = years[i];
      for ( j = 0; j < graph_NHOSTS; j++ )
      {
         if (Y[j][i] > max_Y)
            max_Y = Y[j][i];
         if (Y[j][i] < min_Y)
            min_Y = Y[j][i];
      }
   }
   set_window (min_X, min_Y, max_X, max_Y);  /* region for xsection */
   my_set_viewport(0, 0, v.screenXmax, v.screenYmax);
}  /* init_window_viewport ***************************************************/


/* MISCELLANEOUS <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<*/
void changetextstyle (int height, int width, int fontnum)
/*
DESCRIPTION:
   changes the font; checks for errors that might occur
CALLED BY:
   XYGraph
*/
{
  unsigned char font_arg[20];
  unsigned char *options[NFONTS] =
  {
     "courier", "helv", "tms rmn", "modern", "script", "roman"
  };

  /* build font string */
  strcat( strcat( strcpy(font_arg, "t'"), options[fontnum]), "'");
  sprintf ( font_arg, "h%dw%dp", height, width);
  _setfont(font_arg);

  ErrorCode = _grstatus();                /* check result                */
  if (ErrorCode < _GROK)
    prompt(" SORRY: Graphics Systems Font Error");
} /* changetextstyle ********************************************************/


void line (float X1, float Y1, float X2, float Y2, int thickness)
/*
DESCRIPTION:
   draws a line with the designated thickness
CALLED BY:
   XYGraph
*/
{
  int i;

  for (i = 0; i < thickness; i++ )
  {
```

```
      _moveto((short)X1, (short)Y1--);
      _lineto((short)X2, (short)Y2--);
   }
} /* line ***************************************************************/


/* MAIN ROUTINE TO DRAW GRAPH <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<*/
void XYGraph(int file_index)
/*
DESCRIPTION:
   draws graph of Stem Count vs. Year, DBH vs. Year, etc.
ARGUMENT:
   file_index for labels
CALLED BY:
   control_output_draw
CALLS:
   changetextstyle, my_set_viewport, X_to_screen, Y_to_screen, line
*/
{
  int i, k;
  int ht;       /* character height in pixels */
  int width;    /* character widht  in pixels */
  float graph_X_left,  /* left edge of graph */
      graph_X_right,
      graph_Y_top,     /* top edge of graph */
      graph_Y_bottom;
  int X_heading_loc,   /* locations */
      Y_heading_loc,
      X_label_loc,
      Y_label_loc,
      Y_key_loc,
      key_spacing;
  float incr,          /* incremental amount (tick marks) */
        xstep, ystep;  /* amount to go between increments (tick marks) */
  unsigned short style[6] = { 0xffff, 0x1fff, 0xf0f0, 0xD8D8, 0xFF01, 0xF3C0}; /*line styles*/
  char buffer[10];          /* used in itoa fnc */
  int DECIMAL_SYSTEM = 10; /* used in itoa fnc */

  if (graphics_capable)
     disp_Clear(0);
  /********* calculate locations for headings, etc. in normalized coordinates*/
  ht    = (int)(0.04 * v.screenYmax);  /* 19 pixels in VGA */
  width = (int)(0.03 * v.screenXmax);  /* 19 pixels in VGA */
  graph_X_left   = 0.12 * v.screenXmax;
  graph_X_right  = 0.95 * v.screenXmax;
  graph_Y_top    = 0.1  * v.screenYmax;
  graph_Y_bottom = 0.8  * v.screenYmax;
  X_heading_loc = 0.05  * v.screenXmax;
  Y_heading_loc = 0.04  * v.screenYmax;
  X_label_loc = 0.75 * graph_X_left;
  Y_label_loc = 0.75 * v.screenYmax;
  key_spacing = v.screenXmax/(MAX_HOSTS+1);
  Y_key_loc   = 0.95 * v.screenYmax;

  /********* border **********/
  _setlinestyle(SOLID);
  _setcolor (v.colors[0]);
  _rectangle(_GFILLINTERIOR, 0, 0, v.screenXmax, v.screenYmax);
  _setcolor (v.colors[5]);  /* blue */
  for (i = 0; i < 7; i++)
     _rectangle(_GBORDER, i, i, v.screenXmax-i, v.screenYmax-i);

  /****** heading **********/
  _setcolor(v.max_colors - 1);
  _setgtextvector (1, 0);   /* horizontal text */
  changetextstyle (ht, width, ROMAN);  /* FONT DOESN'T SEEM TO CHANGE */
  _moveto((short)X_heading_loc, (short)Y_heading_loc);
  _outgtext(graphic_outputs[file_index].graph_heading);

  /******** key legend ***********/
  changetextstyle (ht, width, ROMAN);
  for (i = 0; i < graph_NHOSTS; i++)  {
     _setcolor (v.colors[i+1]);
     _moveto((i*key_spacing)+ht, Y_key_loc-ht/2);
     _outgtext(graph_TNAME[i]);
     _setlinestyle(style[i%6]);
     line((i*key_spacing)+  width, Y_key_loc+2*ht/3,
          (i*key_spacing)+3*width, Y_key_loc+2*ht/3, DOUBLE);
  }
   /* reset line style */
  _setcolor (v.max_colors - 1);
  _setlinestyle(SOLID);
```

```
  /********* Y axis label ********************/
  _setgtextvector (0, 1);       /* vertical text */
  changetextstyle ((int)(0.6*ht), (int)(0.6* width), COURIER);
  _moveto(X_label_loc, Y_label_loc);
  _outgtext(graphic_outputs[file_index].label);

  /****** Y axis increments *****************/
  /* Y axis */
  line(graph_X_left, graph_Y_top, graph_X_left, graph_Y_bottom, SINGLE);
  _setgtextvector (1, 0);    /* horizontal text */
  changetextstyle ((int)(0.6*ht), (int)(0.6*ht), COURIER);
  ystep = (graph_Y_bottom - graph_Y_top) / 2;
  incr = graph_Y_bottom;
  for (i=0 ; i < 3; ++i)
  { /* tick marks */
    line(graph_X_left-ht/2, incr, graph_X_left, incr, SINGLE);
    if (i == 0)
    {
       _moveto ((short)(graph_X_left-2*ht), (short)incr);
       _outgtext("0");
    }
    else if (i == 1)
    {
       itoa((int)max_Y/2, buffer, DECIMAL_SYSTEM);
       _moveto ((short)(graph_X_left-3*ht), (short)(incr-0.3*ht));
       _outgtext(buffer);
    }
    else
    {
       itoa((int)max_Y, buffer, DECIMAL_SYSTEM);
       _moveto ((short)(graph_X_left-3*ht), (short)(incr-0.4*ht));
       _outgtext(buffer);
    }
    incr -= ystep;
  }

  /********* X axis labels *********/
  /* X axis */
  line(graph_X_left, graph_Y_bottom, graph_X_right, graph_Y_bottom, SINGLE);
  xstep = (graph_X_right-graph_X_left) / (year_count-1);
  incr = graph_X_left;
  _setgtextvector (0, 1);       /* vertical text */
  changetextstyle (ht/2, ht/2, COURIER);
  for (i=0 ; i<year_count; ++i)
  {  /* tick marks */
     line (incr, graph_Y_bottom, incr, graph_Y_bottom+ht/2, SINGLE);
     _moveto((short)incr, (short)(graph_Y_bottom + 3*ht));
     itoa(years[i], buffer, DECIMAL_SYSTEM);
     _outgtext(buffer);
     incr += xstep;
  }

  /******** Draw Graph **************/
  _setviewport (0, 0, v.screenXmax, v.screenYmax);        /* Open port to full screen        */
  my_set_viewport(graph_X_left, graph_Y_top, graph_X_right, graph_Y_bottom);

  for (i=0 ; i<year_count-1; ++i)
     for (k = 0; k < graph_NHOSTS; k++)  /* MUST READ THIS FROM .D1S FILE! */
     {
        _setcolor (v.colors[k+1]);
        _setlinestyle(style[k%6]);
        line (X_to_screen(years[i]),   Y_to_screen(Y[k][i]),
              X_to_screen(years[i+1]), Y_to_screen(Y[k][i+1]), DOUBLE);
     }
  _setlinestyle(SOLID);          /* reset line style */
  _setcolor(v.max_colors - 1);         /* color to white */
} /*  XYGraph ************************************************************/
```

```
/**************** QUIT VARIABLES & FUNCTIONS HEADER FILE ************/
/* USER INTERFACE variables */
extern boolean EDITS_MADE,
   CHOICE,
   EXIT_NOW;

/* FILE ACCESS variables */
extern char  forestfile_name[9],
       long_forestfile_name[13];


/* FUNCTION PROTOTYPES */
/* routines using CSCAPE function calls */
void verify_choice ( char msg1[50], char msg2[50] );
int forestfile_edit (void);

/* no CSCAPE */
void save_forest (char file_name[13]);
void delete_output_after_save(void);
```

```
/************************* QUIT ****************************/
/* Microsoft C 6.0 includes */
#include <stdio.h>
#include <string.h>
/* CSCAPE 3.2 includes */
#include <cscape.h>
#include <popdecl.h>
/* define file */
#include "defines.h"
/* variables and functions for quit routine */
#include "quit.h"


int exit_routine (VOID *sdata, int idata)
/*
DESCRIPTION:
  exits program after possibly saving file
CALLS:
  verify_choice, save_forest, forestfile_edit, delete_output_after_save
CALLED BY:
  framer menu QUIT
NOTES:
  routine has 5 exit points
*/
{
  int blank_entered_count;

  EXIT_NOW = FALSE;
  if (EDITS_MADE == TRUE)
  {
     CHOICE = TRUE;
     verify_choice ( "SAVE File before EXIT?",
                     "WARNING: Edited data may be lost");
     if (CHOICE == TRUE)
     {
        blank_entered_count = 0;
        while (EXIT_NOW == FALSE)
        {
            /* if user has hit ESC 2 times */
            if ((blank_entered_count > 1) && (strcmp(" ", forestfile_name)==0))
            {
                beep();
                prompt ("     WARNING: File Not Saved");
                EXIT_NOW = TRUE;
                delete_output_after_save();
                return (-1);
            }
            else if (strcmp(" ", forestfile_name)==0)     /* user has not entered name */
            {
                if (blank_entered_count > 0)
                   beep();
                blank_entered_count++;
                low_message("No name entered.  Enter file name or press ESC twice to exit without saving
file.");
                (void) forestfile_edit ();
                low_message(NULL);
            }
            else if ((access(long_forestfile_name, 0)) == 0) /* file exist? */
            {
                 blank_entered_count = 0;
                CHOICE = FALSE;
                verify_choice ( "OVERWRITE existing File?",
                                "WARNING: File exists on disk");
                if (CHOICE == TRUE)
                {
                   /* overwrite file */
                   save_forest (long_forestfile_name);
                   EXIT_NOW = TRUE;
                   delete_output_after_save();
                   return (-1);
                }
                else
                {
                   low_message("Enter NEW disk file name");
                   (void) forestfile_edit ();
                   low_message(NULL);
                }
            }
            else
            {
                /* not overwriting file */
                save_forest (long_forestfile_name);
```

```
                EXIT_NOW = TRUE;
                delete_output_after_save();
                return (-1);
             }
          } /* while */
       }
       else
       {
          EXIT_NOW = TRUE;
          delete_output_after_save();
          return (-1);
       }
    }
    else
    {
       /* no editing done; file not saved; so exit */
       EXIT_NOW = TRUE;
       delete_output_after_save();
       return (-1);
    }
} /* exit_routine ********************************************************/
```

```
/************** INSTALLATION PROGRAM FOR STAND-DAMAGE MODEL ******************/
/*        Unzips and Installs damage files onto hard disk  */
/* Borland C 3.1 includes */
#include <stdio.h>
#include <string.h>     /* strcpy */
#include <conio.h>      /* clrscr */
#include <process.h>    /* system call to dos */
#include <stdlib.h>     /* system call to dos */
#include <dir.h>        /* getdisk, findfirst */
#include <dos.h>        /* sleep */
#include <ctype.h>      /* toupper */

#define heading; printf("\n************** Stand-Damage Model Installation *********\n\n");
#define MIN_STORAGE_NEEDED  650  /* hard disk storage needed in KBytes */
#define CR                  0xD
#define ESC                 0x1B
#define BACKSPACE           0x8
#define BACKSLASH           0x5C
#define FORWARDSLASH        0x2F
#define SPACE               0x20
#define DELETE              0x153
#define TIME                2.5   /* sleep time in seconds */
#define SUCCESS             0
#define FAILURE             -1    /* used with system calls */
#define NUMBER_OF_FILES     8
#define VGA_MONOCHROME      0
#define COLOR               1
#define COLOR_BAK           GREEN
#define COLOR_FORE          WHITE
#define TRUE                1
#define FALSE               0
/* #define TEST */


/* VARIABLES  ********************************************************************/
int key_stroke;  /* keystroke entered */
int system_call_result;
char install_drive[4] = "A:\\";
char zip_file_name[13] = "model.pkd";
char zip_exe_name[13] = "zdamage.exe";
char      zip_cmd[9] = "zdamage";       /* zip command */
char target_drive[4] = "C:\\";
char target_default_path[MAXPATH] = "DAMAGE";
char target_full_path[MAXPATH];
char target_dir[25];
char starting_path[MAXPATH];
int screen_color = COLOR;  /* VGA_MONOCHROME or COLOR */


/* FUNCTION PROTOTYPES ***********************************************************/
int getkey_stroke(void);
void welcome(void);
void beep(void);
void exit_program(void);
void get_target_drive (void);
void get_install_drive(void);
void target_drive_existence (char drive[2]);
void install_drive_existence(char drive[2]);
void get_base(void);
void get_directory(void);
void new_directory (void);
void existing_path(void);
void insure_path(void);
int unzip_damage (void);
void check_drive_storage(char drive[2]);
char *get_current_directory (char *path);
void reset_starting_path(void);
void installation_complete(void);
void installation_not_done (void);
int is_damage_installed (void);
void get_monitor_type (void);
void get_directory_heading  (int show_damage_string);
int valid_char (int c);


void beep (void)
/*
DESCRIPTION:
  make noise for error
CALLED BY:
  welcome, get_target_drive, target_drive_existence, get_install_drive,
  install_drive_existence, check_drive_storage, get_base, insure_path, get_directory
```

```
*/
{
    sound (200);
    delay (200);
    nosound ();
} /* beep ****************************************************************/


void exit_program (void)
/*
DESCRIPTION:
  program termination, return to original path
CALLED BY:
  welcome, get_target_drive, get_install_drive, install_drive_existence,
  get_base, insure_path, get_directory
*/
{
  textbackground(0);
  textcolor(7);
  clrscr();
  printf("Install aborted\n");
  if (0 != chdir(starting_path))
     printf("\n   WARNING: Unable to return to original path\n");
  exit (0);
} /* exit_program *********************************************************/


int getkey_stroke(void)
/*
DESCRIPTION:
  waits for and returns the next keystroke input
CALLED BY:
  welcome, get_target_drive, get_base, insure_path
NOTES:
  keystrokes that the Rom-BIOS describes with extended codes
  are returned as integers > 255
*/
{
   int c ;

   c = getch() ;           /* get single ASCII character */
   if(c == 0)              /* is it a non-ASCII extended code? */
     c = 0x100 + getch() ; /*   yes - use 256 + scan code */
   return(c) ;
} /* getkey_stroke ********************************************************/


char *get_current_directory (char *path)
/*
DESCRIPTION:
  gets current directory
CALLED BY:
  welcome
*/
{
  strcpy (path, "X:\\");
  path[0] = 'A' + getdisk();
  getcurdir(0, path + 3);
  return(path);
} /* get_current_directory ************************************************/


void welcome (void)
/*
DESCRIPTION:
  introduces user to DAMAGE MODEL installation program
CALLED BY:
  main
CALLS:
  getkey_stroke, exit_program, get_current_directory
*/
{
  clrscr();
  printf("            ****************************************\n");
  printf("            *    STAND-DAMAGE MODEL Version 1.1    *\n");
  printf("            *          USDA Forest Service         *\n");
  printf("            *             Morgantown, WV           *\n");
  printf("            ****************************************\n\n\n\n");
  printf("  INSTALL will load the STAND-DAMAGE MODEL files onto your PC.\n\n");
  printf("  650K of available storage space is needed on the target disk.\n\n");
  printf("  This INSTALL program must be run to prepare the software for use.\n\n");
  printf("<Enter> to continue                    ESC to quit\n");
```

```
  key_stroke = getkey_stroke();
  while ((key_stroke != CR) && (key_stroke != ESC))
  {
     beep();
     key_stroke = getkey_stroke();
  }

  printf("    \n\nOne Moment Please...\n");
  /* get initial path so after exiting program, path can be reset */
  get_current_directory(starting_path);

  if (key_stroke == ESC)  /* if user pressed ESCAPE, exit program */
     exit_program ();
} /* welcome ************************************************************/


void get_install_drive (void)
/*
DESCRIPTION:
  allows user to enter installation drive
CALLED BY:
  main
CALLS:
  install_drive_existence, getkey_stroke, exit_program
NOTES:
  recursive
*/
{
  clrscr();
  heading;
  printf("    What drive is the installation program being run from?\n\n");
  printf("    Choose letter A or B\n\n\n");
  printf("<Enter> to continue (A is the default)         ESC to quit\n");

  key_stroke = getkey_stroke();
  switch (key_stroke)  {
     case ESC: exit_program ();
        break;
     case CR:
     case 65:case 97: install_drive_existence("A");break;  /* A */
     case 66:case 98: install_drive_existence("B");break;  /* B */
     case 67:case 99: install_drive_existence("C");break;
     case 68:case 100:install_drive_existence("D");break;
     case 69:case 101:install_drive_existence("E");break;
     case 70:case 102:install_drive_existence("F");break;
     case 71:case 103:install_drive_existence("G");break;
     case 72:case 104:install_drive_existence("H");break;
     case 73:case 105:install_drive_existence("I");break;
     case 74:case 106:install_drive_existence("J");break;
     case 75:case 107:install_drive_existence("K");break;
     case 76:case 108:install_drive_existence("L");break;
     case 77:case 109:install_drive_existence("M");break;
     case 78:case 110:install_drive_existence("N");break;
     case 79:case 111:install_drive_existence("O");break;
     case 80:case 112:install_drive_existence("P");break;
     case 81:case 113:install_drive_existence("Q");break;
     case 82:case 114:install_drive_existence("R");break;
     case 83:case 115:install_drive_existence("S");break;
     case 84:case 116:install_drive_existence("T");break;
     case 85:case 117:install_drive_existence("U");break;
     case 86:case 118:install_drive_existence("V");break;
     case 87:case 119:install_drive_existence("W");break;
     case 88:case 120:install_drive_existence("X");break;
     case 89:case 121:install_drive_existence("Y");break;
     case 90:case 122:install_drive_existence("Z");break;
     default: beep ();
        get_install_drive (); /* recursive call */
        break;
  }
}  /* get_install_drive ***************************************************************/


void install_drive_existence (char drive[2])
/*
DESCRIPTION:
  makes sure that chosen drive exists
CALLED BY:
  get_install_drive
CALLS:
  get_install_drive "pseudo recursion"
SAMPLE CALL
  install_drive_existence ("H");
```

```
NOTES:
  getcurdir normally used to get current directory, we use it to return error
    if drive does not exist
*/
{
   int drive_num;   /* drive_num: 1-drive A, 2-drive B, etc */
   char path[MAXPATH];

   printf("    \n\nOne Moment Please...\n");
   drive_num = (unsigned)(drive[0] - 'A') + 1;
   strcpy (path, "\\");
   /* does drive exist? */
   if (0 == getcurdir(drive_num, path))
   {
      strcpy(install_drive, drive);
      strcat(install_drive, ":\\");
   }
   else /* drive not found */
   {
      beep ();
      printf("\n\n\n   An ERROR occurred!   %s is not a valid drive", drive);
      sleep (TIME);
      get_install_drive ();
   }
}/* install_drive_existence ************************************************/


void get_target_drive (void)
/*
DESCRIPTION:
  allows user to enter target drive
CALLED BY:
  main
CALLS:
  target_drive_existence, getkey_stroke, exit_program
NOTES:
  recursive
*/
{
  clrscr();
  heading;
  printf("    Where should the Model be installed?\n\n");
  printf("    Choose drive letter: C, D, E, F, G, H, A, or B, etc.\n\n\n");
  printf("<Enter> to continue (C is default drive)              ESC to quit\n");

  key_stroke = getkey_stroke();
  switch(key_stroke)  {
     case ESC: exit_program ();
        break;
     case 65:case 97: target_drive_existence("A");break;  /* A */
     case 66:case 98: target_drive_existence("B");break;  /* B */
     case CR:case 67:case 99: target_drive_existence("C");break;
     case 68:case 100:target_drive_existence("D");break;
     case 69:case 101:target_drive_existence("E");break;
     case 70:case 102:target_drive_existence("F");break;
     case 71:case 103:target_drive_existence("G");break;
     case 72:case 104:target_drive_existence("H");break;
     case 73:case 105:target_drive_existence("I");break;
     case 74:case 106:target_drive_existence("J");break;
     case 75:case 107:target_drive_existence("K");break;
     case 76:case 108:target_drive_existence("L");break;
     case 77:case 109:target_drive_existence("M");break;
     case 78:case 110:target_drive_existence("N");break;
     case 79:case 111:target_drive_existence("O");break;
     case 80:case 112:target_drive_existence("P");break;
     case 81:case 113:target_drive_existence("Q");break;
     case 82:case 114:target_drive_existence("R");break;
     case 83:case 115:target_drive_existence("S");break;
     case 84:case 116:target_drive_existence("T");break;
     case 85:case 117:target_drive_existence("U");break;
     case 86:case 118:target_drive_existence("V");break;
     case 87:case 119:target_drive_existence("W");break;
     case 88:case 120:target_drive_existence("X");break;
     case 89:case 121:target_drive_existence("Y");break;
     case 90:case 122:target_drive_existence("Z");break;
     default: beep ();
        get_target_drive (); /* recursive call */
        break;
  }
}  /* get_target_drive ******************************************************/
```

```
void target_drive_existence (char drive[2])
/*
DESCRIPTION:
  makes sure that chosen drive exists
CALLED BY:
  get_target_drive
CALLS:
  get_target_drive  "pseudo recursion", check_drive_storage
SAMPLE CALL
  target_drive_existence ("H");
NOTES:
  getcurdir normally used to get current directory, we use it to return error
    if drive does not exist
*/
{
   char path[MAXPATH];
   int drive_num; /* drive_num: 1-drive A, 2-drive B, etc */

   printf("    \n\nOne Moment Please...\n");
   drive_num = (unsigned)(drive[0] - 'A') + 1;
   strcpy (path, "\\");
   /* does drive exist? */
   if (0 == getcurdir(drive_num, path))
   {
      strcpy(target_drive, drive);
      strcat(target_drive, ":\\");
      check_drive_storage (drive);
   }
   else /* drive not found */
   {
      beep ();
      printf("\n\n\n   An ERROR occurred!   %s is not a valid drive", drive);
      sleep (TIME);
      get_target_drive ();
   }
}/* target_drive_existence ************************************************/


void check_drive_storage(char drive[2])
/*
DESCRIPTION:
  make sure there is enough space on the disk chosen for installation
CALLED BY:
  target_drive_existence
CALLS:
  get_target_drive  "pseudo recursion"
*/
{
   float total_K;       /* total storage on hard disk(K) */
   float K_per_sector;  /* Kbytes per sector */
   int drive_num;
   struct dfree disk_table;  /* dfree declared in dos.h */

   drive_num = (unsigned)(drive[0] - 'A') + 1;
   getdfree(drive_num, &disk_table);

   /* convert bytes to Kbytes */
   K_per_sector = disk_table.df_bsec * 0.000977; /*  1/1024 */
   /*    K = (Kbytes/sector) * (sectors/cluster) * clusters */
   total_K = K_per_sector * disk_table.df_sclus * disk_table.df_avail;

   if (total_K < MIN_STORAGE_NEEDED)
   {
      beep ();
      printf("\n     WARNING: Not enough storage to install on drive %s\n", drive);
      printf("              Only %dK available\n", (int)total_K);
      sleep (4);
      get_target_drive ();
   }
}  /* check_drive_storage ************************************************/


void get_base (void)
/*
DESCRIPTION:
  determine directory base where DAMAGE MODEL should be installed
CALLED BY:
  main
CALLS:
  existing_path (recursive), get_base (recursive), getkey_stroke, exit_program
*/
{
```

```
   clrscr();
   heading;
   printf("     Choose a base path for Model installation.\n\n");
   printf("     Default base: %s\n\n\n", target_drive);
   printf("     E - Enter E to select an Existing directory.\n\n\n");
   printf("<Enter> to continue (default base is %s)         ESC to quit\n", target_drive);

   key_stroke = getkey_stroke();
   switch (key_stroke)
   {
      case ESC: exit_program ();
       break;
      case CR: strcpy(target_full_path, target_drive);
       break;
      case 69:  /* E chosen */
      case 101: /* e chosen */
       existing_path ();
       break;
      default: beep ();
       get_base (); /* recursive call */
       break;
   }
}  /* get_base ***************************************************************/


void existing_path (void)
/*
DESCRIPTION:
  user enters existing path
CALLED BY:
  get_base
CALLS:
  existing_path  (recursive)
NOTES:
  chdir normally used to change directory, we use it to return error
    if path does not exist
*/
{
  char target_temp_path[MAXPATH];

  clrscr();
  heading;
  printf("\n    Enter existing base path where Model is to be installed.\n\n\n");
  printf("                                          Ctrl-Break to quit\n");
  printf("Path  %s", target_drive);  scanf ( "%s", target_dir );
  strcpy(target_temp_path, target_drive);
  strcat(target_temp_path, target_dir);

  if (0 == chdir(target_temp_path))
  {
     strcpy(target_full_path, target_temp_path);
     strcat(target_full_path, "\\");
  }
  else /* invalid directory */
  {
     beep();
     printf("\n   An ERROR occurred!: Invalid pathname entered\n");
     sleep (TIME);
     existing_path ();
   }
}  /* existing_path ***********************************************************/


/*****************************/
void get_directory   (void)
/*****************************
DESCRIPTION:
  determine working directory where DAMAGE MODEL should be installed
*/
{
  int c, string_len=0;
  int last_c = 0;  /* char entered before c */
  int x, y;
  char buffer[81];
  int STOP = FALSE;
  int DEFAULT_PATH = TRUE;

  y = 7;                          /* C:\ */
  x = 20 + strlen(target_full_path) - 3;
  get_directory_heading (TRUE);
  gotoxy(x, y);
  while ((STOP != TRUE) && (string_len < 8)) {
```

```
c = getkey_stroke();
/* no extended ASCII or nonalphanumeric */
if(((c < 128) || (c==DELETE)) && (valid_char(toupper(c))))  {
   switch(c)  {
   case ESC: exit_program ();
             break;
   case CR: if (string_len == 0) {
               if (last_c == DELETE) {
                  /* only try to create directory if > "C:\" */
                  if (strlen(target_full_path) > 3) {
                     printf("\n\n  One Moment Please...\n");
                     /* strip \ off the end of path if it exists */
                     if (target_full_path[strlen(target_full_path) - 1] == '\\')
                         target_full_path[strlen(target_full_path) - 1] = '\0';
                     if (0 != mkdir(target_full_path))  {
                        printf("\n     WARNING: Directory already exists\n");
                        sleep (TIME);
                     }
                  }
                  else {
                     printf("\n\n  Using existing path...\n");
                     sleep (TIME);
                  }
                  STOP = TRUE;
               }
               else {
                  if (DEFAULT_PATH)
                     strcat(target_full_path, target_default_path);
                  /* only try to create directory if > "C:\" */
                  if (strlen(target_full_path) > 3) {
                     printf("\n\n  One Moment Please...\n");
                     /* strip \ off the end of path if it exists */
                     if (target_full_path[strlen(target_full_path) - 1] == '\\')
                         target_full_path[strlen(target_full_path) - 1] = '\0';
                     if (0 != mkdir(target_full_path))  {
                        printf("\n     WARNING: Directory already exists\n");
                        sleep (TIME);
                     }
                  }
               }
            }
            else  {  /* string_len > 0 */
               printf("\n\n  One Moment Please...\n");
               strcat(target_full_path, buffer);
               if (0 != mkdir(target_full_path))  {
                  printf("\n  WARNING: Directory exists or illegal path name.\n");
                  sleep (TIME);
               }
            }
            STOP = TRUE;
            break;
   case DELETE:
            if(string_len == 0) {
               printf("        ");    /* blank out string */
               gotoxy(x, y);
            }
            break;
   case BACKSPACE:
            DEFAULT_PATH = FALSE;
            if (string_len == 0) {
               printf("        ");    /* blank out string */
               gotoxy(x, y);
            }
            else {
               string_len--;
               buffer[string_len] = '\0';
               gotoxy(x, y);
               printf("        ");
               gotoxy(x, y);
               printf("%s", buffer);
            }
            break;
   default: DEFAULT_PATH = FALSE;
            if (string_len == 0) {
               /* blank out string */
               printf("        ");
               gotoxy(x, y);
            }
            printf("%c", c);
            buffer[string_len] = c;
            string_len++;
```

```
                buffer[string_len] = '\0';
                if (string_len == 8) {
                   printf("\n\n  One Moment Please...\n");
                   strcat(target_full_path, buffer);
                   if (0 != mkdir(target_full_path))  {
                      printf("\n     WARNING: Directory already exists.\n");
                      sleep (TIME);
                   }
                   else
                      sleep (TIME);
                }
                break;
          }
       }
       last_c = c;
    } /* while */
}


/*****************************/
int valid_char   (int c)
/*****************************
DESCRIPTION:
  if valid char, returns it; else return 0
*/
{
   int i;

   /* letters */
   for (i=65; i<91; i++)
      if(c==i)
         return(c);
   /* digits */
   for (i=48; i<57; i++)
      if(c==i)
         return(c);
   if ((c==ESC)||(c==CR)||(c==BACKSPACE)||(c==DELETE))
      return(c);

   /* char not correct */
   beep();
   return(0);
}


/************************************************/
void get_directory_heading  (int show_damage_string)
/************************************************
DESCRIPTION:
  places get_directory heading on display
*/
{
  clrscr();
  heading;
  printf("  Press <Enter> for default, delete for existing path, or type new directory.\n\n");
  printf("  Default Path  %sDAMAGE  (up to 8 characters)  ESC to quit\n", target_full_path);
  if (show_damage_string)
     printf("      New Path  %sDAMAGE", target_full_path);
  else
     printf("      New Path  %s", target_full_path);
}


void insure_path (void)
/*
DESCRIPTION:
  make sure user wants destination path
CALLED BY:
  main
CALLS:
  insure_path  (recursive), getkey_stroke, exit_program
*/
{
  clrscr();
  heading;
  printf("     Please confirm this configuration.\n\n");
  printf("         Install disk:   %s\n", install_drive);
  printf("     Destination path:   %s\n\n", target_full_path);
  if (screen_color == COLOR)
     printf("         Monitor type:   Color or Monochrome\n\n");
  else
     printf("         Monitor type:   Grey-scale VGA Monochrome\n\n");
```

```
  printf("<Enter> to continue                          ESC to quit\n");

  key_stroke = getkey_stroke();
  switch (key_stroke)
  {
     case ESC: exit_program ();
      break;
     case CR:  /* user wants to continue */
      break;
     default: beep ();
      insure_path (); /* recursive call */
      break;
  }
}  /* insure_path ************************************************************/


int unzip_damage (void)
/*
DESCRIPTION:
  unZIP damage zip files, rename zip file to executable;
  if monochrome, rename "default.mon" file to "default", erase default.mon
  rename executable back to zip file
RETURNS:
  system_call_result
CALLED BY:
  main
*/
{
  char dos_unzip_cmd[80];       /* DOS "unZIP" string */
  char dos_rename_cmd[80];      /* DOS "rename" string */
  char dos_copy_cmd[80];        /* DOS "copy"  string */
  char dos_erase_cmd[80];       /* DOS "erase" string */

  clrscr();
  heading;
  printf("     Model is being installed in designated directory.\n\n");
  printf("                  PLEASE STANDBY ..........\n\n\n");
  printf("     Designated path  %s\n\n\n", target_full_path);

  /* DOS system call to rename zip file to executable */
  sprintf (dos_rename_cmd, "rename %s%s %s", install_drive, zip_file_name, zip_exe_name);
  system_call_result = system(dos_rename_cmd);
  if (FAILURE == system_call_result)
  {
     printf("ERROR: unable to rename zip file to executable\n");
     return(FAILURE);
  }

  /* unzip damage files */
  sprintf(dos_unzip_cmd, "%s%s %s", install_drive, zip_cmd, target_full_path);
  system_call_result = system(dos_unzip_cmd);
  if (FAILURE == system_call_result)
  {
     printf("ERROR: unable to unzip damage files\n");
     return(FAILURE);
  }

  /* add \ to the end of path if it is not found */
  if (target_full_path[strlen(target_full_path) - 1] != '\\')
     strcat (target_full_path, "\\");

  if (screen_color == VGA_MONOCHROME)
  {
     printf("Copying VGA-Monochrome default settings:\n");
     sprintf(dos_copy_cmd, "copy %sdefault.mon %sdefault",
      target_full_path, target_full_path);
     system_call_result = system(dos_copy_cmd);
     if (FAILURE == system_call_result)
     {
      printf("ERROR: unable to copy default.mon to default\n");
      return(FAILURE);
     }
  }

  /* erase default.mon */
  sprintf(dos_erase_cmd, "erase %sdefault.mon", target_full_path);
  system_call_result = system(dos_erase_cmd);
  if (FAILURE == system_call_result)
  {
     printf("ERROR: unable to erase default.mon file\n");
     return(FAILURE);
  }
```

```
   sleep (2);
   /* DOS system call to rename executable back to zip file */
   sprintf (dos_rename_cmd, "rename %s%s %s", install_drive, zip_exe_name, zip_file_name);
   system_call_result = system(dos_rename_cmd);
   if (FAILURE == system_call_result)
      printf("ERROR: unable to rename executable back to zip file\n");

   return(SUCCESS);
}  /* unzip_damage ************************************************************/


void reset_starting_path (void)
/*
DESCRIPTION:
  return user to original path
CALLED BY:
  main
*/
{
  if (0 != chdir(starting_path))
     printf("\n   WARNING: Unable to return to original path\n");
} /* reset_starting_path ***************************************************/


int is_damage_installed (void)
/*
DESCRIPTION:
  locates all damage files; returns error if any file is not found
CALLED BY:
  main
*/
{
  struct ffblk ffblk;      /* file directory info declared in dir.h */
  char damage_path[80];    /* path to damage file */
  int result;              /* findfirst result (0 :success, -1 :failure) */
  int index;
  char damage_files[NUMBER_OF_FILES][13];  /* damage files to be installed */

  result = SUCCESS;
  index = 0;
  strcpy (damage_files[0], "damage.exe");
  strcpy (damage_files[1], "help");
  strcpy (damage_files[2], "default");
  strcpy (damage_files[3], "species");
  strcpy (damage_files[4], "intro");
  strcpy (damage_files[5], "help.idx");
  strcpy (damage_files[6], "roman.fon");
  strcpy (damage_files[7], "courb.fon");

  while ((result == SUCCESS) && (index < NUMBER_OF_FILES))
  {
     strcpy (damage_path, target_full_path);
     /* add \ to the end of path if it is not found */
     if (target_full_path[strlen(target_full_path) - 1] != '\\')
      strcat (damage_path, "\\");
     strcat (damage_path, damage_files[index]);

     /* is damage file in targeted path? */
     result = findfirst (damage_path, &ffblk, 0);
     index = index+1;
  }
  return (result);
}  /* is_damage_installed ***************************************************/


void installation_complete (void)
/*
DESCRIPTION:
  how to execute DAMAGE MODEL
CALLED BY:
  main
*/
{
  /* strip \ off the end of path if it exists */
  if (target_full_path[strlen(target_full_path) - 1] == '\\')
      target_full_path[strlen(target_full_path) - 1] = '\0';
  clrscr();
  heading;
  printf("The Stand-Damage Model version 1.1 has been installed successfully.\n\n");
  printf("To execute Model, change to directory %s,\n", target_full_path);
  printf("   then type DAMAGE\n\n");
} /* installation_complete*************************************************/
```

```
void installation_not_done (void)
/*
DESCRIPTION:
  inform user we are unable to install DAMAGE MODEL
CALLED BY:
  main
CALLS:
  getkey_stroke
*/
{
  clrscr();
  heading;
  printf("  ERROR: Unable to install Model\n\n");
  printf("  To start the installation procedure again, type %sINSTALL\n\n", install_drive);
} /* installation_not_done*************************************************/


void get_monitor_type (void)
/*
DESCRIPTION:
  allows user to choose between VGA monochrome OR color or monochrome
CALLED BY:
  main
CALLS:
  getkey_stroke, exit_program
*/
{
  clrscr();
  heading;
  printf("    Do you wish to use a Grey-scale VGA monochrome monitor? (Y or N)\n\n");
  printf("    (This screen type is common with laptop and notebook computers.)\n\n\n\n\n");
  printf("<Enter> to continue (N is default)                    ESC to quit\n");

  key_stroke = getkey_stroke();
  switch (key_stroke)
  {
     case ESC: exit_program ();
      break;
     case 121: /* y */
     case 89: screen_color = VGA_MONOCHROME;  /* Y */
      break;
     case CR:
     case 110: /* n */
     case 78:  /* N */
     default: screen_color = COLOR;
      break;
  }
}  /* get_monitor_type ******************************************************************/


int main()
{
#ifndef TEST
   welcome ();

   get_install_drive();
   get_target_drive ();
#else
   strcpy(target_drive, "D:\\");
   strcpy(target_full_path, target_drive);
#endif
   get_base ();
   get_directory ();
   get_monitor_type ();
   insure_path ();
   system_call_result = unzip_damage ();
   reset_starting_path ();

   clrscr();
   /* check to insure installation is successful */
   if ((system_call_result == SUCCESS) && (is_damage_installed() == SUCCESS))
      installation_complete ();
   else /* error occurred */
      installation_not_done ();
   exit(0);
   return(0);
}/* END of main */
```

Racin, George; Colbert, J. J. 1994. **Guide to the Stand-Damage Model interface management system.** Gen. Tech. Rep. NE-209. Radnor, PA: U.S. Department of Agriculture, Forest Service, Northeastern Forest Experiment Station. 149 p.

Describes the Gypsy Moth Stand-Damage Model interface management system. Management of stand-damage data made it necessary to define structures to store data and provide the mechanisms to manipulate these data. The software is used to manipulate files, graph and manage outputs, and edit input data. The interface was built using pop-up windows, menuing systems, text editing and validation, mouse support, and context-sensitive help. The interface is written in the C language for DOS microcomputers.

Keywords: Software; forest growth; simulation; computer models

Send requests for publications to:
USDA Forest Service
Publications Distribution
359 Main Road
Delaware, OH 43015

Printed on Recycled Paper

Headquarters of the Northeastern Forest Experiment Station is in Radnor, Pennsylvania. Field laboratories are maintained at:

Amherst, Massachusetts, in cooperation with the University of Massachusetts

Burlington, Vermont, in cooperation with the University of Vermont

Delaware, Ohio

Durham, New Hampshire, in cooperation with the University of New Hampshire

Hamden, Connecticut, in cooperation with Yale University

Morgantown, West Virginia, in cooperation with West Virginia University

Orono, Maine, in cooperation with the University of Maine

Parsons, West Virginia

Princeton, West Virginia

Syracuse, New York, in cooperation with the State University of New York, College of Environmental Sciences and Forestry at Syracuse University

University Park, Pennsylvania, in cooperation with The Pennsylvania State University

Warren, Pennsylvania

"Caring for the Land and Serving People Through Research"